Spis treści rocznika

11–12 1988

P.1877/88
BK3

informatyka

NETBIOS
Maszyny MIMD i SIMD
Język VHDL
Rozmowa z komputerem

WYDAWNICTWO NOT SIGMA

Cena 500 zł

informatyka

11–12 1988

Nr 11–12

Miesięcznik Rok XXIII

Listopad–Grudzień 1988

Organ Komitetu
Naukowo-Technicznego NOT
ds. Informatyki

KOLEGIUM REDAKCYJNE:
Mgr Jarosław DEMINET,
dr inż. Wacław ISZKOWSKI,
mgr Teresa JABŁOŃSKA
(sekretarz redakcji),
Władysław KLEPACZ
(redaktor naczelny),
dr inż. Marek MACHURA,
dr inż. Wiktor RZECZKOWSKI,
mgr inż. Jan RYŻKO,
mgr Hanna WŁODARSKA,
dr inż. Janusz ZALEWSKI
(zastępca redaktora naczelnego).

PRZEWODNICZĄCY
RADY PROGRAMOWEJ:

Prof. dr hab.
Juliusz Lech KULIKOWSKI

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 01-517 Warszawa, ul. Mickiewicza 18 m. 17, tel. 39-14-34

RSW „PRASA-KSIĄŻKA-RUCH"
PRASOWE ZAKŁADY GRAFICZNE
ul. Dworcowa 13, 85-950 BYDGOSZCZ
Zam. 4024/88. E-4
Obj. 8,0 ark. druk. Nakład 9050 egz.

ISSN 0542-9951. INDEKS 36124

Cena egzemplarza 500 zł

NACZELNA ORGANIZACJA TECHNICZNA
WYDAWNICTWO
CZASOPISM I KSIĄŻEK TECHNICZNYCH
SIGMA

00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE **Strona**

NETBIOS – zasada działania i sposób użytkowania (1)
Adam Tucholski 1
Uproszczone specyfikowanie systemów współbieżnych (2)
Leslie Lamport 7
Rozmowa z komputerem
Agnieszka Mykowiecka, Anna Ostaszewska 11
Język opisu sprzętu VHDL – podstawowe mechanizmy (1)
Henryk Gizdoń, Adam Pawlak, Włodzimierz Wrona 15
AMOK – mikrokomputerowy system doradczy projektanta i analityka systemów komputerowych
Tadeusz Czachórski, Marek Kowalówka, Zdzisław Szczerbiński, Andrzej Wilk 19
DAGGER – program utrzymania plików
Piotr Fuglewicz, Roman Topczyński 23
System komputerowy Elwro 800 (2)
Wojciech Cellary, Jerzy Kręglewski, Ruta Maćkowiak 26
Maszyny i algorytmy równoległe (2). Maszyny MIMD i SIMD
Maciej M. Sysło 30
Turbo C. Przerwania, sygnały, powroty
Jan Bielecki 35
Przegląd sieci lokalnych dla IBM PC
Piotr Strutyński 37

ZE ŚWIATA 40

Perspektywy rozwoju układów ASIC
Rynek drukarek komputerowych w 1988 roku
Program bibliotekarza Ady
Środowisko programowe Ady
Teoretyczne podstawy informatyki
Kto jest kim w IFIP. Wolfgang D. Ehrenberger

RECENZJE 51

Data & Knowledge Engineering

TERMINOLOGIA III okł.

Terminologia systemów operacyjnych (3)

W NAJBLIŻSZYCH NUMERACH

- **Bogumiła Pawlak charakteryzuje zasady działania metaasemblerów mikroprogramów oraz rozwiązań metaasemblera MIC, opracowanego w Instytucie Informatyki Politechniki Warszawskiej**
- **Krystyna Kandziora-Blauman i Jerzy Kisilewicz omawiają sprzętową realizację poszczególnych poziomów protokołu liniowego sieci komputerowych**
- **Mieczysław Kłopotek opisuje zagadnienia komputerowej generacji mowy z tekstu pisanego i realizację pakietu syntezy mowy TMK, przeznaczonego dla mikrokomputerów typu IBM PC/XT/AT**
- **Zbigniew Struk i Ewa Zabża-Tarka prezentują prosty system zarządzania bibliograficzną bazą danych dla mikrokomputera IBM PC/XT**
- **Jan Bielecki kontynuuje praktyczne wskazówki i podaje przykłady zastosowania rozwiązań języka programowania Turbo C na mikrokomputerach klasy IBM PC**
- **Mariusz Kuc opisuje dwa pakiety bibliotek procedur Turbo Pascala – Power Tools Plus, wersja 2 i Turbo Extender**

ADAM TUCHOLSKI
Instytut Łączności
Zakład Teleinformatyki
Warszawa

POLITECHNIKA ŚLĄSKA Biblioteka Główna Gliwice
P. 1877/88

NETBIOS
– zasada działania i sposób użytkowania (1)

Wkrótce po upowszechnieniu się komputerów osobistych powstała potrzeba dzielenia przez nie wspólnych zasobów. Było to wynikiem zarówno czynników ekonomicznych, takich jak wysokie koszty dobrych drukarek czy dysków stałych, jak również, co istotniejsze, potrzeba podziału wspólnej bazy danych. Powstało wówczas wiele sieci, z których szczególnie upowszechniły się w Polsce: Transnet i D-Link. Mają one jednak pewne wady, a mianowicie:

- brak standardowego sprzężenia programowego,
- obsługa zleceń sieci poniżej systemu operacyjnego, na poziomie BIOS-u.

Z pewnym opóźnieniem na sytuację rynkową zareagowała firma IBM wraz z firmą Microsoft, wprowadzając sieć IBM PC Network.

Podstawową cechą tej sieci jest użycie standardowego, sprzętowo niezależnego, sprzężenia między adapterem sieci a DOS-em, w postaci zbioru procedur nazwanego NETBIOS. Dostarcza on wielu gotowych mechanizmów, takich jak funkcje tworzenia sesji i wymiany informacji między użytkownikami sieci, a także – funkcje zarządzania siecią. W ten sposób procedury obsługujące wyższe poziomy protokołu nie muszą zawierać informacji, na jakim sprzęcie pracują. Umożliwia to stworzenie jednego programu sieciowego, obsługującego wszystkie rodzaje sieci.

Inna ważna cecha to umieszczenie najwyższego poziomu procedur obsługi sieci w systemie operacyjnym DOS. W ten sposób zapewniono realizowanie zleceń do sieci przez DOS komputera, na którym znajduje się udostępniony zasób. System operacyjny zachowuje więc pełną kontrolę nad tym, co dzieje się na jego komputerze, i zapobiega kolizjom podczas jednoczesnego dostępu kilku programów do tego samego zasobu. Tak więc możliwe jest zastosowanie bardziej subtelnych mechanizmów ochrony w powstałej wielodostępnej bazie danych i ograniczenie wielkości chronionego przed niepożądanym dostępem obszaru do katalogu, pliku czy wręcz jednego bajtu.

Ta sieć stała się wkrótce standardem, czego skutkiem było powstanie wielu wartościowych programów aplikacyjnych, działających w tym środowisku. Aby mogły one działać w innych sieciach, wystarczy zasymulować funkcje realizowane normalnie przez NETBIOS, za pomocą specjalnego oprogramowania – emulatora NETBIOS-a. Takie emulatory znalazły się w ofercie firm produkujących karty adapterów sieci komputerowych. Oferuje je również producent sieci D-Link.

Mgr inż. ADAM TUCHOLSKI ukończył studia na Wydziale Elektroniki Politechniki Warszawskiej. Zajmuje się problematyką dotyczącą sieci komputerowych typu lokalnego i rozległego, a w szczególności publiczną siecią teledacyjną SYNKOM z komutacją kanałów.

ADAPTER IBM PC NETWORK

Firma IBM oferuje dla sieci IBM PC Network własne rozwiązanie adaptera, dla którego funkcje NETBIOS-a są zaprogramowane w pamięci ROM. NETBIOS jest rozszerzeniem oprogramowania systemowego BIOS, na tyle zintegrowanym z adapterem sieci, że z pozycji programisty można go traktować jako element adaptera.

Przy projektowaniu sieci występuje zawsze problem podziału siedmiu warstw oprogramowania (według architektury OSI/ISO) między adapter sieci a komputer. Dla sieci IBM PC Network wprowadzono następujący podział funkcji: adapter realizuje warstwy 1–5 według architektury OSI/ISO (warstwy ukierunkowane na komunikację wraz z warstwą sesji), a pozostałe warstwy są realizowane w komputerze macierzystym (ang. *host*) jako rozszerzenie systemu operacyjnego (np. przy wykorzystaniu programu IBM PC LAN).

Adapter sieci IBM PC Network jest przystosowany do pracy z szerokopasmowym kablem TV i składa się z części cyfrowej i analogowej – modemu (rys.). Modem dołączony do kabla koncentrycznego realizuje funkcje warstwy najniższej modelu OSI/ISO (fizycznej[1]). Koprocesor 82586 w połączeniu ze sprzęgiem szeregowym uzupełnia realizację dwóch najniższych warstw architektury OSI/ISO (fizycznej i łącza danych). Trzy wyższe warstwy (sieciowa, transportowa, sesji) są realizowane przy wykorzystaniu lokalnego procesora Intel 80188 oraz pamięci stałych ROM na płycie adaptera. Lokalny procesor 80188 zawiera dwa niezależne kanały DMA, programowane zegary i sterownik przerwań. Komunikacja między komputerem macierzystym a adapterem odbywa się przez układ sprzęgający. Adapter sieci wykorzystuje kanały DMA do przesyłania danych (w dwóch kierunkach) między dwoma rejestrami danych typu FIFO układu sprzęgającego a buforami w lokalnej pamięci. Procesor realizuje cztery przerwania zewnętrzne. Dwa pochodzą od układu sprzęgającego, jedno od sprzęgu szeregowego oraz jedno od koprocesora 82586. Lokalny procesor przygotowuje bufory dla koprocesora 82586 oraz wykonuje programy obsługi protokołu komunikacyjnego.

Pamięć dynamiczna RAM (16 KB) jest używana przez procesor jako pamięć notatnikowa (ang. *scratch memory*), stos i bufory informacji nadawanej i odbieranej. Pamięć stała ROM (32 KB) zawiera programy dla procesora 80188. Pamięć stała PROM (32 B) zawiera informacje o adresie fizycznym adaptera sieci (różnym dla każdej karty). Ten adres identyfikuje się ze stałą nazwą węzła. Pamięć BIOS ROM jest bezpośrednio dostępna dla komputera macierzystego i realizuje sprzężenia programowe między sterownikiem sieci i komputerem macierzystym. Może ona być odłączana przy użyciu przełącznika na płycie adaptera.

CHARAKTERYSTYKA NETBIOSA

NETBIOS daje programiście środki, które umożliwiają wysłanie informacji do odległego komputera, określając specyficzne

[1] Szybkość transmisji wynosi 2 Mb/s, w sieci może pracować 1000 stacji, a maksymalna odległość między stacjami jest ograniczona opóźnieniami i nie powinna przekraczać 5 km.

cechy sieci lokalnej w formie standardu. Możliwe jest wiarygodne przesyłanie połączeniowe. Jeśli dane są zagubione lub ośrodek transmisji uszkodzony, NETBIOS sygnalizuje to kodem błędu. Wiadomości przesyłane przy użyciu datagramów przechodzą bezpośrednio do warstwy łącza, nie wykorzystując w pełni możliwości warstwy sieciowej i transportowej. Przesyłanie bezpołączeniowe stosuje się głównie dla wiadomości typu rozgłaszanego (ang. *broadcast*) z możliwością zagubienia informacji.

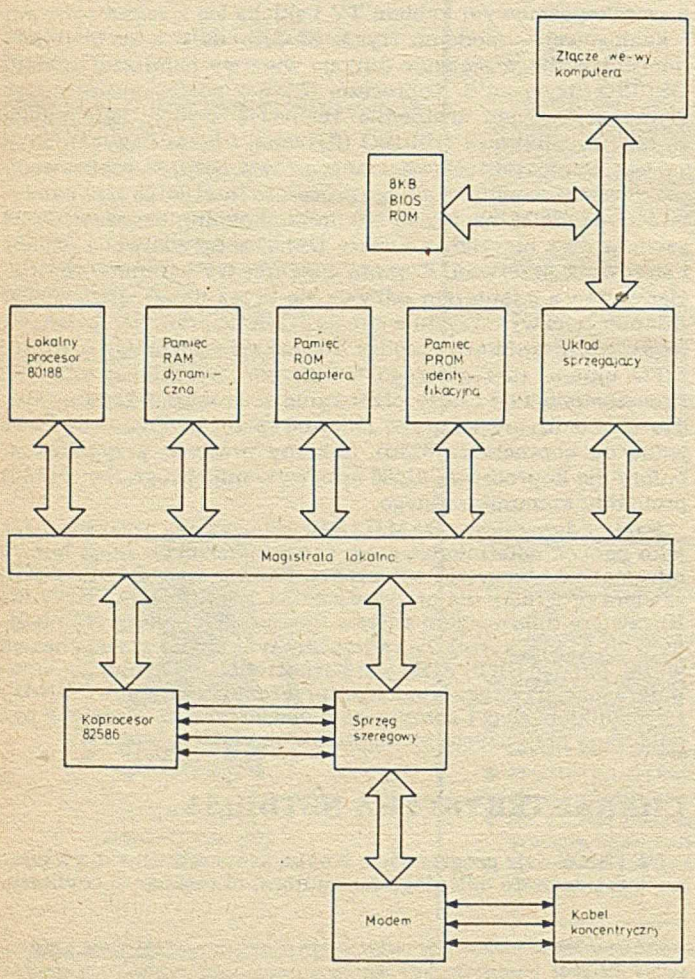
NETBIOS umożliwia komunikowanie się w sieci przy użyciu nazw. Każdemu adapterowi sieci można przypisać szesnaście różnych nazw oraz jedną stałą nazwę węzła, która jest zawsze obecna (zapamiętana w PROM-ie). Stała nazwa składa się z 10 bajtów zerowych uzupełnionych numerem identyfikacyjnym węzła. Wadą NETBIOS-a jest to, że nie ma wbudowanych mechanizmów ochrony danych.

W NETBIOS-ie powiązano realizację trzech warstw architektury OSI/ISO (sieciowej, transportowej i sesji), gdzie każda warstwa komunikuje się z przyległymi.

Warstwa sieciowa odpowiada za zestawianie połączeń wirtualnych i komutację pakietów po optymalnej trasie. Gdy warstwa sieciowa otrzymuje jednostkę danych z warstwy łącza danych, to określa, czy pakiet jest datagramem, czy też należy do połączenia wirtualnego, i przesyła go warstwie transportowej do dalszego opracowania.

Warstwa transportowa zamyka przepływ informacji na trasie dwupunktowej (ang. *point-to-point*). Ta warstwa odpowiada za transmisję danych w trybie potwierdzeń oraz sterowanie przepływem. Uwalnia ona wyższe warstwy od organizowania niezawodnego i efektywnego przesyłania danych. Przesyła wiadomości warstwie sesji, korzystając z usług warstwy sieciowej.

Warstwa sesji organizuje i synchronizuje (ze względu na różne pojemności buforów) wymianę informacji między użytkownikami oraz strukturalizuje w ustalony sposób wymianę danych. Sesje są zestawiane między nazwami (przechowywanymi w odpowiednich tablicach) reprezentującymi poszczególne procesy.



Budowa adaptera sieci

W warstwie sesji zrealizowano koncepcję przyporządkowania adapterom sieci wielu nazw. W tej warstwie są interpretowane polecenia przekazywane od użytkownika w formie bloków NCB (ang. *network control blocks*).

Korzystając z NETBIOS-a w trybie połączeniowym należy:
- Nadać danemu komputerowi lokalnemu nazwę, przez którą jest on rozpoznawany w sieci (można też korzystać ze stałej nazwy węzła).
- Nawiązać sesję z inną nazwą w sieci przez zestawienie połączenia logicznego (ta nazwa może pochodzić z tablicy nazw lokalnego bądź odległego adaptera sieci).
- Nadawać i odbierać wiadomości przy użyciu połączenia logicznego (sesji).

Niezależnie od przesyłania połączenia można też korzystać z datagramów.

STEROWANIE PRACĄ SIECI

Polecenia do wykonania są przekazywane przez użytkownika w formie bloków NCB. Zasady współpracy bloku NCB z NETBIOS-em są następujące:
- Należy utworzyć i wypełnić pola bloku NCB. Tworząc nowy NCB należy wyzerować wszystkie pola, a następnie uzupełnić każde pole bloku zgodnie z tabelą 1.
- Zarezerwować niezbędne bufory określone przez pola bloku NCB.
- Upewnić się, czy dla każdego aktywnego bloku NCB zarezerwowano co najmniej 20 bajtów stosu.
- Umieścić adres bloku NCB w parze rejestrów ES:BX i wykonać programowe przerwanie 5CH.
- Blok NCB nie może być zmieniany ani przenoszony podczas realizacji polecenia.
- Po wykonaniu polecenia sterowanie jest przekazywane do programu generującego przerwanie 5CH. Wynik zrealizowanego procesu jest przekazywany zarówno przez rejestr AL, jak i pole NCB_RETCODE, por. tab. 1.

Poniżej podano szczegółowy opis pól bloku NCB.
NCB_COMMAND – bajtowe pole kodu polecenia do wykonania.

Tabela 1. Opis pól bloku NCB

Nazwa pola	Kod	Znaczenie
NCB_COMMAND	DB 00H	Pole polecenia
NCB_RETCODE	DB 00H	Pole kodu powrotnego
NCB_LSN	DB 00H	Pole numeru sesji
NCB_NUM	DB 00H	Numer nazwy użytkownika
NCB_BUFFER ⓐ	DD 00000000H	Adres bufora wiadomości
NCB_LENGTH	DW 0000H	Długość bufora
NCB_CALLNAME	DB 16DUP(0)	Nazwa zadeklarowana lokalnie, bądź w innym węźle sieci; przy przesyłaniu szeregowym dwóch bloków danych (CHAIN SEND) pierwsze 2 bajty określają długość, a następne 4 bajty adres drugiego bufora; pozostałe bajty są zarezerwowane
NCB_NAME	DB 16DUP(0)	Nazwa zadeklarowana lokalnie
NCB_RTO	DB 00H	Wartość przeterminowania (ang. *time-out*) przy odbiorze
NCB_STO	DB 00H	Wartość przeterminowania przy nadawaniu
NCB_POST ⓐ	DD 00000000H	Adres procedury powrotu (znak ⓐ reprezentuje adres fizyczny)
NCB_LANA_NUM	DB 00H	Numer adaptera sieci (wartość 01H jest używana dla drugiego adaptera)
NCB_CMD_CPLT	DB 00H	Pole stanu polecenia
NCB_RESERVE	DB 14DUP (0)	Pole zarezerwowane

Każde polecenie może być wykonywane zarówno w trybie synchronicznym, jak i asynchronicznym. Jeśli najbardziej znaczący bit jest ustawiony na 0, to wybrana jest praca synchroniczna. Pozostałych 7 bitów służy do wybrania odpowiedniego polecenia. Tryb synchroniczny polega na zainicjowaniu polecenia i oczekiwaniu aż NETBIOS zakończy jego wykonywanie. Następnie sterowanie jest przekazywane do programu użytkownika (tzn. do następnej instrukcji po przerwaniu 5CH). W rejestrze AL oraz w polu NCB_RETCODE znajduje się kod powrotny. Tryb asynchroniczny polega na zainicjowaniu polecenia z natychmiastowym powrotem do następnej instrukcji po przerwaniu 5CH. Dla tego rodzaju sterowania wyróżnia się dwa rodzaje kodów powrotnych. Kod natychmiastowy jest udostępniany w rejestrze AL bezpośrednio po zainicjowaniu polecenia. Jeśli jego wartość wynosi 00H, co oznacza prawidłowe wywołanie, to na zakończenie jest udostępniany kod ostateczny. Zakończenie polecenia może się odbywać przez przerwanie lub badanie pola NCB_CMP_CPLT. Jeżeli zawartość pola NCB_POST ⓐ jest różna od zera, to adapter sieci generuje przerwanie pod adres wskazany przez pole NCB_POST ⓐ Kod powrotny znajduje się w polu NCB_RETCODE lub rejestrze AL. Jeśli zawartość NCB_POST ⓐ jest równa zero, to program nie jest przerwany i zakończenie odpowiedniego procesu odbywa się po zbadaniu pola NCB_CMP_CPLT. Jeśli natychmiastowy kod powrotny jest różny od 00H, to nie może być wykonane żadne polecenie, a rodzaj błędu jest sygnalizowany w rejestrze AL (tab. 2).

Tabela 2. Zestawienie kodów powrotnych

Wartość	Znaczenie
00H	Polecenie pomyślnie wykonane
01H	Błędnie zadeklarowana długość bufora przy nadawaniu datagramów lub odczycie stanu
03H	Błędny kod polecenia
05H	Wykonanie polecenia zakończone przeterminowaniem
06H	Dane odebrane niekompletnie
08H	Wybrany numer sesji nie istnieje
09H	Nie ma warunków do zestawienia sesji (chwilowa niedostępność lokalnych zasobów pamięci)
0AH	Sesja zamknięta
0BH	Wykonywanie polecenia odwołane
0DH	Zdublowana nazwa w lokalnej tablicy nazw
0EH	Przepełnienie lokalnej tablicy nazw; zadeklarowanie więcej niż 16 nazw
0FH	Nazwa lokalna nie może być usunięta, ponieważ uczestniczy w sesji; usuwana nazwa jest wyrejestrowana, jednak zajmuje miejsce w tablicy
11H	Tablica lokalnych sesji jest pełna
12H	Sesja nie może być zestawiona (nie wykonano polecenia LISTEN po stronie odbiorczej)
13H	Nieprawidłowo wybrany numer nazwy
14H	Nie można znaleźć nazwy wywoływanej w lokalnej tablicy nazw
15H	Nazwa nieprawidłowo zadeklarowana („*" lub 00H w pierwszym bajcie nazwy) lub nie można jej znaleźć w lokalnej tablicy nazw
16H	Nazwa użyta już w innym adapterze sieci
17H	Nazwa lokalna została usunięta
18H	Awaryjne zakończenie sesji
19H	Wykrycie identycznych nazw przez protokół komunikacyjny (konflikt nazw)
1AH	Otrzymano niestandardowy pakiet (zdalne urządzenie niezgodne z obowiązującym protokołem)
21H	Chwilowa zajętość sprzętu (NETBIOS w trakcie obsługi przerwania)
22H	Przekroczenie maksymalnej liczby poleceń oczekujących na wykonanie
23H	Nieprawidłowo zadeklarowany numer adaptera sieci (różny od 00H lub 01H)
24H	Polecenie zakończone przed próbą jego odwołania
26H	Nielegalna próba odwołania polecenia
4XH	NETBIOS wykrył nieprawidłowy stan sieci (X – oznacza dowolną cyfrę szesnastkową)
50-FEH	Adapter sieci wykrył wewnętrzny błąd
FFH	Polecenie w trakcie realizacji

NCB_RETCODE – bajtowe pole zawierające kod powrotny polecenia. Jeśli kod powrotny jest równy zero, to polecenie zostało wykonane pomyślnie. Inny numer kodu oznacza wykonanie błędne lub niezakończenie polecenia (por. tab. 2). Przy pracy w trybie asynchronicznym bez przerwań, pole NCB_CMP_CPLT (a nie NCB_RETCODE) sygnalizuje wynik realizacji polecenia i zawiera ostateczny kod powrotny. Jeśli polecenie jest w trakcie wykonywania, to pole NCB_CMP_CPLT zawiera wartość 0FFH.

NCB_LSN – bajtowe pole numeru sesji, liczone modulo 254 w przedziale od 1 do 254, określone po pomyślnej realizacji polecenia CALL lub LISTEN. Polecenia SEND i RECEIVE wypełniając to pole wybierają odpowiednią sesję. Polecenie RESET wykorzystuje to pole do określenia maksymalnej liczby obsługiwanych sesji.

NCB_NUM – bajtowe pole numeru nazwy liczone modulo 253 w przedziale od 2 do 254, zapisywane po pomyślnym wykonaniu poleceń ADD NAME i ADD GROUP NAME. Stała nazwa węzła ma przyporządkowany zawsze numer 1. Ten numer (nie nazwa) musi być użyty we wszystkich poleceniach dotyczących przesyłania datagramów i w poleceniu RECEIVE ANY. Polecenie RESET używa tego pola do określenia maksymalnej liczby obsługiwanych bloków NCB.

NCB_BUFFER ⓐ – 4-bajtowe pole adresu bufora, który ma być użyty przy realizacji danego polecenia.

NCB_ LENGTH – 2-bajtowe pole określające długość bufora (w bajtach) przy realizacji poleceń: nadawania, odbioru i odczytu stanu.

NCB_CALLNAME – 16-bajtowe pole określające nazwę partnera w komunikacji. Nazwa może być zadeklarowana w tablicy nazw lokalnego bądź odległego adaptera sieci. Wszystkie bajty tego pola są używane do zdefiniowania nazwy.

NCB_NAME – 16-bajtowe pole określające nazwę zadeklarowaną w lokalnej tablicy nazw. Wszystkie bajty tego pola są używane do zdefiniowania nazwy. Użytkownik może być również rozpoznawany przez stałą nazwę węzła. Stała nazwa węzła składa się z dziesięciu bajtów dwójkowo-zerowych, zakończonych przez 6 bajtów udostępnianych przy realizacji polecenia ADAPTER STATUS.

NCB_RTO – bajtowe pole wykorzystywane przez polecenia CALL i LISTEN do określenia przeterminowania dla wszystkich poleceń RECEIVE związanych z daną sesją. Realizacja polecenia RECEIVE dla danej sesji będzie przerwana, jeśli nie zakończy się pomyślnie przed upływem określonego czasu. Wartość kwantu czasowego wynosi 500 ms. Dla wartości zerowej przeterminowanie jest wyłączone.

NCB_STO – 1-bajtowe pole o przeznaczeniu identycznym jak pole poprzednie, lecz dla polecenia SEND. Przeterminowanie przy nadawaniu zrywa sesję.

NCB_POST ⓐ – 4-bajtowe pole wskaźnika procedury (obsługi przerwania), do której nastąpił skok po wykonaniu polecenia. Do procedury obsługi wchodzi się z zamaskowanymi przerwaniami i priorytetem, jaki został przydzielony adapterowi sieci. Wyjście z procedury następuje przy użyciu instrukcji IRET. To pole jest używane tylko przy pracy asynchronicznej. Jeśli

zawartość pola jest zerowa, to nie bierze ono udziału w przetwarzaniu i program użytkowy sprawdza stan polecenia w polu NCB_CMD_CPLT.

NCB_LANA_NUM – bajtowe pole określające, z którym adapterem sieci użytkownik chce pracować:
00H – pierwszy adapter sieci,
01H – drugi adapter sieci.
Uwaga! Dzięki wbudowanym mechanizmom, NETBIOS umożliwia współpracę sieci różnego rodzaju (np. Arcnet, Ethernet itp.).

NCB_CMD_CPLT – bajtowe pole stanu polecenia. Wartość OFFH sygnalizuje realizację polecenia. Wartość 00H oznacza pomyślne wykonanie polecenia. Każda inna wartość sygnalizuje błąd zgodnie z tab. 2. To pole jest aktywne jedynie wtedy, gdy pole NCB_POST ⓐ jest wyzerowane. W przeciwnym wypadku należy sprawdzić stan polecenia w polu NCB_RETCODE.

NCB_RESERVE – 14-bajtowe pole wykorzystywane przez NETBIOS do tymczasowego przechowywania danych.

POLECENIA NCB

Polecenia bloku NCB sterują pracą adaptera sieci. Są one podzielone na cztery klasy:
- ogólne,
- obsługi nazw,
- obsługi sesji (tryb połączeniowy),
- obsługi datagramów (tryb bezpołączeniowy).

Wszystkie polecenia, oprócz RESET i CANCEL, mogą działać synchronicznie i asynchronicznie.

Polecenia ogólne

Polecenia ogólne są używane do incjowania adaptera sieci, odczytywania stanu oraz kontroli realizacji poleceń.

RESET

Zeruje stan lokalnego adaptera sieci oraz tablice nazw i sesji. Umożliwia zmianę liczby obsługiwanych sesji i bloków NCB w zakresie od 1 do 32. Włączenie zasilania lub podanie wartości zerowych w odpowiednich polach bloku NCB specyfikuje użycie wartości domyślnych, tzn. 6 sesji i 12 poleceń.

CANCEL

Odwołuje realizację polecenia. Adres odpowiedniego bloku NCB znajduje się w polu NCB_BUFFER ⓐ . Odwołanie polecenia SEND powoduje zerwanie sesji.
Uwaga! Realizacja następujących poleceń nie może być odwołana: ADD NAME, ADD GROUP NAME, DELETE NAME, SEND DATAGRAM, SEND BROADCAST DATAGRAM, SESSION STATUS, RESET oraz CANCEL.

ADAPTER STATUS

Podaje informację o aktualnym stanie lokalnego lub odległego adaptera sieci, którego nazwa występuje w polu NCB_CALLNAME. Jeśli w pierwszym bajcie pola NCB_CALLNAME jest znak „*", to otrzymujemy informację o lokalnym adapterze sieci. Informacja jest przesyłana do bufora (o adresie NCB_BUFFER ⓐ i długości NCB_LENGTH). Minimalna długość informacji przesyłanej do bufora stanu wynosi 60 bajtów, maksymalna – 348 bajtów, gdy zadeklarowanych jest 16 nazw. Ogólnie, liczba bajtów w buforze stanu równa się 60 + 18 (X), gdzie X jest liczbą wykorzystywanych nazw.
Dane przesyłane do bufora stanu mają następującą postać:

- Identyfikator węzła – 6 bajtów
bajt 0 – najmniej znaczący
...
bajt 5 – najbardziej znaczący
- Stan przełączników na płycie adaptera sieci – 1 bajt
bity 0-5 – zarezerwowane
bity 6-7 – przełącznik W1/W2 włączony (11) lub wyłączony (00)
- Wynik lokalnego testowania na płycie adaptera sieci po włączeniu zasilania – 1 bajt
- Programowa wersja protokołu komunikacyjnego – 2 bajty
- Statystyka błędów i nasilenia ruchu – 18 bajtów
– przedział czasu objęty sprawozdaniem (w minutach) – 2 bajty; wartość liczona modulo 0FFFFH;
– liczba odebranych błędnych sum kontrolnych CRC – 2 bajty;po osiągnięciu 0FFFFH wartości nie zwiększa się dalej;
– liczba odebranych błędów podczas zestrajania – 2 bajty; po osiągnięciu 0FFFFH wartości nie zwiększa się dalej;
– liczba wykrytych kolizji – 2 bajty; wartość liczona modulo 0FFFFH;
– liczba zerwanych transmisji – 2 bajty; wartość liczona modulo 0FFFFH;
– liczba pakietów prawidłowo wysłanych – 4 bajty; wartość liczona modulo 0FFFFFFFFH;
– liczba pakietów prawidłowo odebranych – 4 bajty; wartość liczona modulo 0FFFFFFFFH;
– liczba retransmisji – 2 bajty; wartość liczona modulo 0FFFFH.
– liczba przypadków, w których odbiornik wyczerpał swoje zasoby – 2 bajty; wartość liczona modulo 0FFFFH.
- Statystyka zasobów adaptera sieci.
– zarezerwowane na pola robocze – 8 bajtów,
– liczba wolnych bloków poleceń – 2 bajty,
– zadeklarowana maksymalna liczba bloków NCB – 2 bajty,
– maksymalna liczba wolnych bloków poleceń – 2 bajty,
– zarezerwowane na pola robocze – 4 bajty,
– liczba aktywnych sesji – 2 bajty; zliczane są tu następujące sytuacje:
CALL w trakcie realizacji,
LISTEN w trakcie realizacji,
sesja zestawiona,
sesja przerwana,
HANG UP w trakcie realizacji,
HANG UP zakończone,
– zadeklarowana maksymalna liczba aktywnych sesji – 2 bajty,
– całkowita liczba możliwych sesji – 2 bajty.
- Liczba nazw zadeklarowanych w lokalnej tablicy nazw – 2 bajty
- Tablica lokalnych nazw – 16 pozycji po 18 bajtów każda.

Pierwsze 16 bajtów każdej pozycji w tablicy określa nazwę, a ostatnie dwa bajty określają stan. Pierwszy bajt jest równy numerowi nazwy, a drugi bajt określa stan, przy czym istotne są tylko trzy najmniej znaczące bity i najbardziej znaczący bit tego bajtu (N = 0 – nazwa wyłączna, N = 1 – nazwa grupowa). Pozostałe bity są zarezerwowane i mogą być różne od zera:
NXXXX000 – nazwa w trakcie rejestracji,
NXXXX100 – nazwa zarejestrowana,
NXXXX101 – nazwa wyrejestrowana, lecz nie usunięta z lokalnej tablicy nazw,
NXXXX110 – wykrycie podwójnej nazwy,
NXXXX111 – wykrycie podwójnej nazwy, która jest równocześnie wyrejestrowana.
Po pomyślnym wykonaniu polecenia udostępniana jest długość informacji zapisanej do bufora stanu (pole NCB_LENGTH). Realizacja polecenia może być przerwana wskutek przeterminowania.

Obsługa nazw

Polecenia obsługi nazw umożliwiają nadanie nazwy procesowi użytkowemu. Nazwa może być wyłączna lub grupowa. Stała nazwa węzła jest zastrzeżona jako wyłączna. Polecenie RESET usuwa wszystkie nazwy z określonego adaptera sieci, z wyjątkiem stałej nazwy węzła. Następujące nazwy nie mogą być nadawane ani usuwane:
– nazwy zaczynające się od znaku „*" lub 00H,
– nazwy zaczynające się od IBM.

ADD NAME

Wpisuje 16 znaków do lokalnej tablicy nazw. Ta nazwa nie może być użyta przez inny proces użytkowy. Realizacja polecenia rozpoczyna się od sprawdzenia, czy wybrana nazwa była wcześniej zadeklarowana w lokalnej lub odległej tablicy nazw. Jeśli żądana nazwa była zadeklarowana w lokalnej tablicy nazw, to udostępniany jest jej numer oraz kod błędu 0DH (zdublowanie

nazwy w lokalnej tablicy nazw). Sprawdzenie w odległych tablicach nazw polega na wielokrotnym wysłaniu do wszystkich węzłów sieci zapytania o nazwę. Lokalnie przeglądane są odległe tablice nazw i jeśli żaden węzeł nie potwierdzi zapytania o nazwę, to wpisuje się ją do lokalnej tablicy nazw i uznaje za wyłączną. Polecenie powoduje również przypisania nazwie numeru. Numer nazwy jest udostępniany w polu NCB_NUM a następnie wykorzystywany do adresowania datagramów oraz realizacji polecenia RECEIVE ANY.

ADD GROUP NAME

Wpisuje 16 znaków do lokalnej tablicy nazw. Wiele procesów użytkowych może użyć tej samej nazwy grupowej (ang. *group name*). Nazwa grupowa nie może być uprzednio zadeklarowana jako wyłączna. Realizacja polecenia rozpoczyna się od sprawdzenia, czy wybrana nazwa była wcześniej zadeklarowana w lokalnej lub odległej tablicy nazw. Jeśli żądana nazwa była uprzednio zadeklarowana w lokalnej lub odległej tablicy nazw, to jest udostępniany numer oraz kod błędu 0DH (zdublowanie nazwy w lokalnej tablicy nazw). Sprawdzenie w odległych tablicach nazw polega na wielokrotnym wysłaniu do wszystkich węzłów sieci zapytania o nazwę wyłączną. Lokalnie przeglądane są odległe tablice nazw i jeśli żaden węzeł nie potwierdzi zapytania o nazwę wyłączną, to wpisuje się ją do lokalnej tablicy nazw i uznaje za grupową. Polecenie powoduje również przypisanie nazwie numeru. Numer nazwy jest udostępniany w polu NCB_NUM, a następnie wykorzystywany do adresowania datagramów oraz realizacji polecenia RECEIVE ANY.

DELETE NAME

Usuwa nazwę (16 znaków) z tablicy nazw. Jeżeli usuwana nazwa bierze udział w sesji, to udostępniany jest kod błędu 0FH. Realizacja polecenia jest opóźniona do momentu zakończenia sesji lub jej przerwania. Przd usunięciem nazwy należy użyć polecenia HANG UP.

Obsługa sesji

Polecenia obsługi sesji umożliwiają zestawienie połączenia logicznego (sesji), wysłanie i odbiór wiadomości, zakończenie sesji oraz odczytanie stanu sesji. Sesja może być zestawiana z nazwą zadeklarowaną w tablicy lokalnej lub z nazwą zadeklarowaną w odległym adapterze sieci. Nazwy są wykorzystywane tylko do zestawienia połączenia. Po zestawieniu połączenia, w polu NCB_LSN udostępniany jest 1-bajtowy numer sesji, który jest następnie używany przy przesyłaniu połączeniowym zamiast pełnego 16-bajtowego adresu. Dopuszcza się maksymalnie 32 sesje. Ta sama para nazw może dotyczyć więcej niż jednej sesji. Dana sesja może obsługiwać równocześnie nadawanie i odbiór. Polecenie RESET zrywa wszystkie sesje.

CALL

Polecenie zestawia połączenie między lokalną nazwą podaną w polu NCB_NAME oraz drugą nazwą (lokalną lub odległą) podaną w polu NCB_CALLNAME. Nazwa wywoływana jest kolejno poszukiwana w lokalnej tablicy nazw, a jeśli nie jest znaleziona, wysłane jest wielokrotne zapytanie do wszystkich węzłów sieci, które przeglądają lokalne tablice nazw. Po stronie nazwy wywoływanej musi być wygenerowane polecenie LISTEN. Realizacja polecenia może być przerwana wskutek przeterminowania. Pomyślna realizacja polecenia kończy się nawiązaniem połączenia, któremu przydzielony jest tzw. numer sesji (pole NCB_LSN).

LISTEN

Umożliwia zestawienie połączenia między lokalną nazwą wyłączną podaną w polu NCB_NAME oraz drugą nazwą wyłączną (lokalną lub odległą) podaną w polu NCB_CALLNAME. Jeśli na pierwszym miejscu pola NCB_CALLNAME znajduje się znak „*", to połączenie zostanie zestawione z dowolnym węzłem sieci żądającym połączenia. Realizacja polecenia nie może być przerwana wskutek przeterminowania. Pomyślna realizacja polecenia kończy się nawiązaniem połączenia, któremu przydzielany jest tzw. numer sesji (pole NCB_LSN). Jeśli polecenie używa znaku „*", to w polu NCB_CALLNAME udostępniana jest nazwa węzła, z którym zestawiono sesję. Polecenie LISTEN dla określonej nazwy ma wyższy priorytet niż LISTEN dla dowolnej nazwy.

HANG UP

Zamyka połączenie o numerze podanym w polu NCB_LSN. Wszystkie zainicjowane polecenia RECEIVE są zaniechane, zaś realizacja polecenia SEND jest przeprowadzona do końca (jeśli zakończy się przed upływem 20 s).

SEND

Przesyła dane korzystając z numeru sesji (pole NCB_LSN). Dane są pobierane z bufora (o adresie NCB_BUFFER ⓐ i długości NCB_LENGTH). Długość bufora jest ograniczona do 65 535 bajtów. Wykonywanie polecenia może być przerwane wskutek przeterminowania ustalonego przy nawiązywaniu połączenia. Jeśli kilka poleceń SEND oczekuje na wykonanie, to ich realizacja przebiega szeregowo, zgodnie z zasadą FIFO.

CHAIN SEND

Przesyła sekwencyjnie dane korzystając z numeru sesji (pole NCB_LSN). Dane są pobierane kolejno z dwóch buforów i łączone w całość. Pierwszy bufor jest określony tak, jak dla polecenia SEND. Pole NCB_CALLNAME jest wykorzystywane do podania drugiego bufora (tabela 1). Wykonanie połączenia może być przerwane wskutek przeterminowania, ustalonego przy nawiązywaniu połączenia. Jeśli kilka poleceń CHAIN SEND oczekuje na wykonanie, to ich realizacja przebiega szeregowo, zgodnie z zasadą FIFO.

RECEIVE

Odbiera dane korzystając z numeru sesji (pole NCB_NUM). Dane są ładowane do bufora (o adresie NCB_BUFFER ⓐ i długości NCB_LENGTH). Jeżeli długość bufora przygotowanego do odbioru jest niewystarczająca, to udostępniany jest kod błędu 06H (dane odebrane niekompletnie). Można użyć następnego polecenia odbioru dla odzyskania brakującej informacji, lecz przed upływem przeterminowania. Po pomyślnym wykonaniu polecenia udostępniana jest długość danych odebranych (pole NCB_LENGTH). Wykonanie polecenia może być zaniechane wskutek przeterminowania, ustalonego przy nawiązywaniu połączenia. Jeśli kilka poleceń RECEIVE oczekuje na wykonanie, to ich realizacja przebiega stosownie do priorytetu odbioru:
- RECEIVE,
- RECEIVE ANY dla określonej nazwy,
- RECEIVE ANY dla każdej nazwy zadeklarowanej w lokalnej tablicy nazw.

W ramach priorytetu odbiór odbywa się szeregowo zgodnie z zasadą FIFO.

RECEIVE ANY

Zapewnia odbiór danych od każdego, kto ma z nazwą lokalną zestawione połączenie. W tym poleceniu należy używać 1-bajtowego numeru własnej nazwy (pole NCB_NUM) zamiast pełnej 16-bajtowej nazwy. Numer nazwy jest udostępniany po wykonaniu polecenia ADD NAME lub ADD GROUP NAME (pole NCB_NUM). Podanie w polu NCB_NUM wartości 0FFH umożliwia odbiór informacji przesłanej połączeniowo dla dowolnej nazwy w lokalnej tablicy nazw.

Jeżeli długość bufora przygotowanego do odbioru jest niewystarczająca, to udostępniany jest kod błędu 06H. Dla odzyskania brakującej informacji można użyć następnego polecenia RECEIVE, lecz przed upływem przeterminowania po stronie nadawcy. Po pomyślnym wykonaniu polecenia udostępniane są informacje w odpowiednich polach bloku NCB: NCB_LENGTH, NCB_LSN, NCB_NUM. Wykonywanie polecenia nie może być przerwane wskutek przeterminowania ustalonego przy nawiązywaniu połączenia. Jeśli kilka RECEIVE oczekuje na wykonanie, to ich realizacja przebiega według zasady opisanej powyżej.

SESSION STATUS

Zapisuje do bufora (o adresie NCB_BUFFER ⓐ i długości NCB_LENGTH) stan wszystkich aktywnych sesji dla nazwy z lokalnej tablicy nazw podanej w polu NCB_NAME. Podanie znaku „*" w pierwszym bajcie pola NCB_NAME specyfikuje wszystkie nazwy z tablicy. Minimalna postać informacji przesyłanej do bufora stanu wynosi 4 bajty. Dane przesyłane do bufora mają następującą postać:

- numer nazwy, dla której przygotowuje się sprawozdanie – 1 bajt,
- liczba sesji skojarzonych z daną nazwą – 1 bajt,
- liczba zaległych poleceń RECEIVE DATAGRAM oraz RECEIVE BROADCAST DATAGRAM oczekujących na realizację – 1 bajt,
- liczba zaległych poleceń RECEIVE ANY oczekujących na realizację – 1 bajt,
- Informacje dotyczące stanu sesji – 36 bajtów dla każdej sesji
  - numer lokalnej sesji – 1 bajt,
  - stan lokalnej sesji – 1 bajt, opisany następująco:
    LISTEN w trakcie realizacji – 01H,
    CALL w trakcie realizacji – 02H,
    sesja zestawiona – 03H,
    HANG UP w trakcie realizacji – 04H,
    HANG UP zakończone – 05H,
    sesja zerwana – 06H,
  - nazwa lokalna – 16 bajtów,
  - nazwa odległa – 16 bajtów,
  - liczba zaległych poleceń RECEIVE oczekujących na realizację – 1 bajt,
  - liczba zaległych poleceń SEND i CHAIN SEND oczekujących na realizację – 1 bajt.

Po pomyślnym wykonaniu polecenia udostępniana jest długość informacji zapisanej do bufora stanu (pole NCB_LENGTH). Realizacja polecenia nie może być przerwana wskutek przeterminowania.

Obsługa datagramów

Polecenia obsługi datagramów umożliwiają przesyłanie datagramów do adresatów o nazwie wyłącznej, nazwie grupowej, wszystkich nazwach w węzłach oraz – odbiór datagramowy od adresatów określonych w ten sam sposób. Przy przesyłaniu datagramów wiadomość nigdy nie jest potwierdzana przez odbiorcę, a więc może ulec zagubieniu. Przesyłana wiadomość jest ograniczona do 512 bajtów. Przy nadawaniu dane są pobierane z bufora (o adresie NCB_BUFFER ⓐ i długości NCB _LENGTH), a przy odbiorze przesyłane do bufora. Po pomyślnym wykonaniu polecenia udostępniane są dodatkowe informacje w odpowiednich polach bloku NCB: NCB_LENGTH, NCB_CALLNAME. Jeżeli długość bufora przygotowanego do odbioru jest niewystarczająca, to udostępniany jest kod błędu 06H. Pozostałe dane nie odebrane są stracone.

SEND DATAGRAM

Przesyła datagram do adresata określonego nazwą wyłączną, nazwą grupową – we własnej lub odległej stacji. Polecenie wykorzystuje do adresowania numer własnej nazwy (pole NCB_NUM) oraz pełny adres określający adresata wiadomości (pole NCB_CALLNAME). Datagram jest przesyłany do określonego węzła z wyszczególnioną nazwą, równą numerowi nazwy adresata. Polecenie RECEIVE DATAGRAM po stronie odbiorczej sprawdza, czy w otrzymanym datagramie jest wyspecyfikowany numer nazwy oraz porównuje go z własnym. Jeśli datagram nie ma wyspecyfikowanego numeru nazwy, to jest odbierany dowolnym poleceniem RECEIVE DATAGRAM.

SEND BROADCAST DATAGRAM

Przesyła datagram do wszystkich węzłów sieci. Polecenie wykorzystuje do adresowania numeru własnej nazwy (pole NCB_NUM).

RECEIVE DATAGRAM

Odbiera datagramy od dowolnego węzła sieci, który przesyłał wiadomość.

dokończenie na s. 36

Nowe książki

J.S. Haugdahl, Inside NETBIOS, Architecture Technology Corp., Minneapolis (MN), 1986, ISBN 0-939405-01-6

Książka jest pogłębioną prezentacją systemu NETBIOS, opracowanego pierwotnie jako sprzęg programowy dla karty adaptera sieciowego do sieci IBM PC Network. Publikacja jest przeznaczona dla zaawansowanych użytkowników NETBIOS-a i ma charakter opisu technicznego (ang. *technical reference*). Na początku omówiono w niej pochodzenie NETBIOS-a w perspektywie historycznej oraz jego miejsce w rodzinie sieci komputerowych na tle siedmiowarstwowego modelu odniesienia ISO/OSI. W głównej części książki przedstawiono dwie najważniejsze implementacje systemu NETBIOS dla sieci firmy IBM: pierwotną dla sieci PC Network i emulator dla sieci Token-Ring. Oprócz koniecznego zilustrowania podobieństw i różnic tych dwóch implementacji firmy IBM w tej części porównano je z implementacjami innych firm. Co więcej, przedyskutowano także rozwiązania innych sieci, np.sprzęg programowy APPC (ang. *advanced program-to-program communication*) sieci Token-Ring oraz sieci firmy Microsoft. Ważnym elementem książki jest wskazanie na ograniczenia wszystkich sieci tego rodzaju, spowodowane oparciem obsługi plików na jednozdaniowym systemie operacyjnym PC-DOS.

Najważniejszą częścią książki jest opis poleceń NETBIOS-a i sposobu korzystania z bloku NCB (ang. *network control block*). Po dokładnej prezentacji struktury bloku NCB, omówiono wszystkie polecenia, dzieląc je na cztery kategorie – ogólną, obsługi nazw, obsługi sesji i obsługi datagramów – podając także systematyczny wykaz możliwych błędów. W dodatku, może nazbyt skromnie, zawarto istotne informacje dotyczące tworzenia programów aplikacyjnych opartych na NETBIOS-ie. Szczegółowo omówiono programowanie bram (ang. *gateways*) do innych sieci i – rozproszonych baz danych. Dobór takich właśnie przykładów jest bardzo zręczny, ze względu na gwałtowny rozwój zastosowań sieciowych w tych obu kierunkch. Przy ich omawianiu wskazano jednak na znaczne niedostatki użytkowania tą metodą innych sieci i sieciowych baz danych. Książkę uzupełniono spisem kilkunastu wartościowych pozycji bibliograficznych dotyczących NETBIOS-a.

J.S. Haugdahl, Inside the Token-Ring, North-Holland, Amsterdam, 1986, ISBN 0-444-70139-7.

Sieć lokalna IBM Token-Ring zgodna z amerykańską normą IEEE 802.5 i międzynarodową ISO 8802.5 ma szanse – z uwagi na właściwości dogodne do zastosowań przemysłowych – zdominować rynek urządzeń sieciowych przeznaczonych dla przemysłu. W książce omówiono od wewnątrz tę sieć, stanowiącą jeden z najbardziej interesujących technicznie wyrobów lat osiemdziesiątych.

Pierwszy rozdział zawiera ogólny przegląd sieci Token-Ring, przedstawia jej opracowanie, eksploatację, zapewniane usługi i kierunki rozwoju. W rozdziale drugim omówiono rozwiązania sprzętowe sieci, tj. system okablowania zastosowany przez firmę IBM, jednostkę dostępu do ośrodka (MAU, ang. *media access unit*), adapter dla komputera PC i zaprojektowany specjalnie dla sieci Token-Ring zestaw mikroukładów VLSI. W rozdziale trzecim przedstawiono kolejno NETBIOS, włącznie z blokiem NCB (ang. network control block), zasady komunikacji asynchronicznej oraz protokoły na poziomie łącza logicznego i łącza fizycznego. Rozszerzeniem tego rozdziału jest rozdział czwarty, który zawiera omówienie: programu obsługi adaptera PC, poleceń NETBIOS-a i sprzęgu APPC (ang. *advanced program-to program communication*).

Usługi zapewniane przez sieci Token-Ring komputerom PC przedstawiono w rozdziale piątym. Należą do nich, m. in.: komunikacja asynchroniczna, dostęp do dużych komputerów sieci SNA (ang. *system network architecture*), połączenia wzajemne z siecią IBM PC Network, obsługa plików i obsługa druku.

dokończenie na s. 51

LESLIE LAMPORT
Digital Equipment Corporation
Systems Research Center
Palo Alto, stan Kalifornia
USA

Uproszczone specyfikowanie systemów współbieżnych (2)

W drugiej części artykułu przedstawiono trzy inne specyfikacje: dokładniejszą specyfikację automatu z wodą sodową i dwie specyfikacje bazy danych.

INNA SPECYFIKACJA AUTOMATU Z WODĄ SODOWĄ

Na wydruku 1 przedstawiono inną specyfikację automatu z wodą sodową, używając specjalnego języka przypominającego zwykły deklaracyjny język programowania. Deklaracje poprzedzone klauzulą **interface procedures** określają specyfikację sprzężenia, która jest pominięta. Deklaracje z klauzulą **var** określają zakres wartości przybieranych przez **x** i **y**. Nawiasy kątowe obejmują operacje, które są niepodzielne (atomowe). Instrukcja γ wykonuje akcję złożoną z akcji sprzęgającej **deposit_coin** (wrzucenie monety) i akcji ustawienia zmiennej **y** na wartość wrzuconej monety. Klauzula **only if**, wprowadzona jedynie na użytek tej instrukcji, znaczy że akcja może wystąpić tylko wtedy, gdy wartość **x** + **y** po nowym przypisaniu wynosi co najwyżej 50. Tak więc, klauzula **only if** zapobiega możliwości wrzucenia monety półdolarowej po wrzuceniu ćwierćdolarowej.

```
interface procedures deposit_coin ... ;
                     dispense_soda ... ;
var x: {0,25,50};
    y: {25,50};
begin α: ⟨x := 0⟩;
      β: while ⟨x < 50⟩
           do γ: ⟨y := deposit_coin only if x + y_new ≤ 50⟩;
              δ: ⟨x := x + y⟩
           od;
      ε: ⟨dispense_soda⟩
end
```

Wydruk 1. Inna specyfikacja automatu z wodą sodową

Wydruk 1 zawiera program. Jak go się interpretuje jako specyfikację w metodzie aksjomatu przejść?

Aby zinterpretować ten program jako specyfikację w metodzie aksjomatu przejść, należy opisać funkcje stanu, akcje oraz pozostałe elementy specyfikacji. Wewnętrznymi funkcjami stanu są zmienne **x** i **y** oraz dodatkowa funkcja stanu, o nazwie **pc**, opisująca stan sterowania programem. Funkcje stanu sprzężenia są wyspecyfikowane domyślnie (np. niejawnie) w pominiętej części klauzuli **interface procedures**. Funkcja stanu **x** może przybierać wartości 0, 25 i 50 a funkcja stanu **y** – wartości 25 i 50. Funkcja stanu **pc** przybiera wartości: α, β, γ, δ, ε. Wartości początkowe funkcji **x** i **y** są nieokreślone, a wartość początkowa **pc** wynosi α, wskazując, że sterowanie jest początkowo ustawione na instrukcję α. Jest pięć akcji, wszystkie objęte nawiasami kątowymi i opatrzone etykietami α, ..., ε. Akcje γ i ε są akcjami sprzężenia wykonywanymi, odpowiednio, przez otoczenie i przez system; pozostałe są akcjami wewnętrznymi. Aksjomaty przejść dla tych akcji specyfikują następujące dopuszczalne zmiany funkcji stanu.

α: Ta akcja może wystąpić tylko wtedy, gdy **pc** ma wartość α. Zmienia wartość **x** na 0, wartość **pc** na β, a wartość **y** pozostawia niezmienioną.

β: Może wystąpić tylko wtedy, gdy **pc** = β. Jeżeli **x** $<$ **50**, to zmienia wartość **pc** na γ; w przeciwnym razie zmienia wartość **pc** na ε. Wartości **x** i **y** pozostawia bez zmian.

γ: Reprezentuje akcję użytkownika polegającą na wrzuceniu monety. Ustawia wartość **y** na równą wartości wrzuconej monety, która musi być monetą ćwierć- lub półdolarową (ponieważ **y** może równać się tylko 25 lub 50) i pozostawia wartość **x** bez zmian. Ta akcja może wystąpić tylko wtedy, gdy **pc** = γ a nowa wartość **y** spełnia nierówność **x** + **y** $\leqslant$ **50**.

δ: Może wystąpić tylko wtedy, gdy **pc** = δ. Ustawia wartość **x** na poprzednią wartość powiększoną o poprzednią wartość **y**. Pozostawia wartość **y** bez zmiany i ustawia wartość **pc** na β.

ε: Może wystąpić tylko wtedy, gdy **pc** = ε. Ustawia wartość **pc** na α i pozostawia wartości **x** i **y** bez zmian. Ta akcja reprezentuje wydanie puszki z wodą sodową.

W tej specyfikacji **x** *i* **y** *wyglądają jak zwykłe zmienne programu, lecz* **pc** *wydaje się dziwne. Czy jest jakaś zasadnicza różnica między funkcją stanu* **pc** *a funkcjami stanu* **x** *i* **y**?

Nie. Aby opisać wykonanie programu napisanego w języku deklaracyjnym, należy określić, jak zmienia się sterowanie programem i jak zmieniają się wartości zmiennych. Programista nie może jawnie odwoływać się do „licznika rozkazów", ponieważ jego wartość jest zarówno częścią stanu programu jak i wartością zwykłej zmiennej. W celu przedstawienia stanu obliczeń, programista ma często do wyboru, czy użyć dodatkowej zmiennej czy bardziej złożonej struktury sterowania.

Czy każdy program jest specyfikacją?

Tak, program napisany w dowolnym języku programowania można interpretować jako specyfikację metodą aksjomatu przejść. Głównym zadaniem w konstruowaniu kompilatora języka źródłowego na język wynikowy jest przedstawienie w języku wynikowym funkcji stanu specyfikowanych przez program, włącznie z takimi jak **pc** i **stos** (do wywoływania procedur), które nie są jawnie zadeklarowane. Program napisany w języku wysokiego poziomu jest specyfikacją kodu wynikowego generowanego przez kompilator. Jedyną różnicą między programem a specyfikacją wysokiego poziomu jest to, że program jest realizowany przez kompilator bez udziału człowieka.

Jeżeli każdy program jest specyfikacją, to dlaczego nie można wyrażać specyfikacji w zwykłym języku programowania, lecz trzeba używać języków specyfikacyjnych?

Tak można robić, jednak języki programowania są ograniczone przez wymaganie, aby programy były kompilowane na względnie efektywny kod. Ponieważ specyfikacje nie muszą być kompilowane, języki specyfikacyjne umożliwiają prostsze specyfikacje niż mogą być napisane w językach programowania. Ponadto, języki programowania prowadzą do tworzenia zbyt restryktywnych specyfikacji. Przykładowo, w większości języków programowania można łatwo stwierdzić, że jedna akcja musi następować po drugiej, lecz trudno jest wyrazić fakt, że obie akcje mogą być wykonywane w dowolnej kolejności. Takie języki prowadzą do specyfikacji, które niepotrzebnie ograniczają kolejność wykonywania akcji.

Jakich rodzajów konstrukcji można używać w językach specyfikacyjnych, a nie można w językach programowania?

Podstawową konstrukcją języka programowania, służącą do wyrażania jawnych zmian wartości, jest instrukcja przypisania. W języku specyfikacyjnym, instrukcję przypisania można rozszerzyć tak, aby wyrazić dowolną relację między poprzednimi a nowymi wartościami funkcji stanu. Przykładowo, instrukcję γ z wydruku 1 można przedstawić jako następującą relację między poprzednimi a nowymi wartościami zmiennych:

$$y_{new} = \text{deposit_coin} \wedge x_{new} = x_{old} \wedge y_{new} + x_{old} \leqslant 50 \qquad (1)$$

gdzie **deposit_coin** jest pewnym wyrażeniem wiążącym poprzednie i nowe wartości funkcji stanu sprzężenia, która jest zdefiniowana domyślnie przez pominiętą specyfikację sprzężenia.

Zwykła instrukcja przypisania jest szczególnym przypadkiem po relacji, w której nowa wartość zmiennej równa się wartości wyrażenia wiążącego tylko poprzednie wartości zmiennych. Jednakże, istnieją ogólniejsze relacje, jak np.:

$$a_{old} = \sin^2 b_{new} + 3 \cdot \cos b_{new}$$

która wyraża związek między nową wartością zmiennej **b** i poprzednią wartością zmiennej **a**. Taka relacja nie może być wyrażona w języku programowania, ponieważ nie może być skompilowana na efektywny kod. Nie ma żadnego powodu, aby jej nie używać w języku specyfikacyjnym.

Aksjomat przejść dla akcji, który określa zmiany funkcji stanu dopuszczalne dla tej akcji, jest właśnie taką relacją między poprzednimi a nowymi wartościami funkcji stanu. Przykładowo aksjomat przejść dla instrukcji γ otrzymuje się przez połączenie relacji (1), która opisuje dopuszczalne zmiany **x** i **y**, z relacją:

$$pc_{old} = \gamma \wedge pc_{new} = \delta$$

która zapewnia dopuszczalne zmiany **pc** i zawiera wymaganie, że akcja może być wykonana tylko wtedy, gdy wartość początkowa **pc** równa się γ. Program z wydruku 1 można zastąpić zbiorem pięciu aksjomatów przejść o tej postaci. Jednakże, specyfikacja jest bardziej czytelna, jeśli używa się zwykłych konstrukcji języka programowania, takich jak „;" i **while**, aby niejawnie opisać dopuszczalne zmiany **pc**, zamiast jawnego pisania relacji między jego poprzednimi i nowymi wartościami.

Rysunek i program są dwoma różnymi specyfikacjami automatu z wodą sodową. Jaki jest związek między nimi?

Są one równoważne przy założeniu, że nie są uzupełnione o odpowiednie specyfikacje sprzężeń. Inaczej mówiąc, każda z nich jest poprawną realizacją drugiej. Poniżej wykażę, że diagram jest poprawną realizacją programu. Wykazanie prawdziwości w przeciwną stronę wymaga kilku pojęć, które zostaną wprowadzone w kolejnym przykładzie.

```
if pc = α                  → f = I  □
   pc = β or pc = γ        → if x = 0   → f = I   □
                             x = 25  → f = II  □
                             x = 50  → f = III (impossible if pc = γ)
                             fi □
   pc = δ                  → if x + y = 25 → f = II  □
                             x + y = 50 → f = III □
                             x + y = 75 → impossible
                             fi □
   pc = ε                  → f = III
fi
```

Wydruk 2. Definicja funkcji f za pomocą x, y i pc

Interpretacja diagramu jako specyfikacji metodą aksjomatu przejść zakłada istnienie funkcji stanu **f** o pewnych właściwościach. Aby udowodnić, że program spełnia tę specyfikację, należy wykazać istnienie funkcji **f**. Wykonuje się to definiując **f** za pomocą funkcji stanu **x**, **y** i **pc**, których istnienie jest zapewnione przez interpretację programu jako specyfikacji metodą aksjomatu przejść. Najpierw zauważmy, że $x<50$ gdy **pc** $=\gamma$, a $x + y \leqslant 50$, gdy **pc** $=\delta$. Dowodzi się tego wykazując, że te dwie asercje są prawdziwe początkowo i pozostają prawdziwe przy każdej akcji. Wartość funkcji **f** jest zdefiniowana wyrażeniem z wydruku 2, zapisanym przy użyciu konstrukcji Dijkstry[1] **if**. Ostatecznie wykażę, że dla takiej definicji funkcji **f**, każda akcja dozwolona przez specyfikację według programu odpowiada akcji (zmianie f) dozwolonej przez specyfikację według diagramu. Czytelnik może sprawdzić, że akcje α, β i γ nie zmieniają funkcji **f**. Przykładowo akcja α może być wykonana tylko wtedy, gdy **pc** $=\alpha$ (wówczas **f** = **I**), a po jej wykonaniu **pc** $=\beta$ i **x** = **0** (tzn. **f** pozostaje równe **I**). Czytelnik może również sprawdzić, że wykonanie akcji γ odpowiada *wrzuceniu monety ćwierćdolarowej* lub *wrzuceniu monety półdolarowej* z diagramu. Przykładowo, załóżmy że akcja γ jest wykonywana dla **x** = **25**. Ponieważ może to być wykonane tylko wtedy, gdy **pc** $=\gamma$, wynika stąd że początkowo **f** = **II**. Ze specyfikacji γ wynika, że dla wartości początkowej **x** = **25** taka akcja może zmienić wartość **x**, **y** i **pc** tylko ustawiając **y** na 25 i **pc** na δ, co oznacza że **f** = **III**. Taka zmiana wartości **f** z **II** na **III** jest zapewniana przez akcję *wrzucenie monety ćwierćdolarowej* z diagramu. Czytelnik może sprawdzić, że druga możliwość, tzn. wykonanie akcji dla **x** = **0**, również powoduje zmianę wartości **f** dozwoloną przez akcje *wrzucenie monety ćwierćdolarowej* lub *wrzucenie monety półdolarowej* z diagramu. Podobnie, wykonanie akcji ε zmienia **f** w sposób dopuszczalny dla przedstawionej na diagramie akcji *wydanie puszki*.

Czy to jest wszystko, czego należy dowieść?

Nie dowiedziono jeszcze, że program stanowi poprawną realizację sprzężenia z diagramu. Wymaga to jednak wykazania, że akcje **deposit_coin** i **dispense_soda** są poprawnymi realizacjami odpowiednich akcji przedstawionych na diagramie. Nie można tego jednak uczynić, ponieważ nie ma specyfikacji sprzężenia ani w programie, ani w diagramie.

Jak można sformalizować nieformalne rozumowanie przeprowadzone w dowodzie?

Niech *wektor stanu* dla specyfikacji według programu będzie trójką dopuszczalnych wartości **x**, **y** i **pc**, a wektor stanu dla specyfikacji według diagramu – dopuszczalną wartością funkcji **f** (**I**, **II** lub **III**). Aby zdefiniować funkcję **f** za pomocą **x**, **y** i **pc**, należy zdefiniować odwzorowanie **F** wektorów stanu programu na wektory stanu z diagramu. Przykładowo, **F**(**0**, **25**, δ) = **II** znaczy, że **f** = **II**, gdy **x** = **0**, **y** = **25** i **pc** = δ.

Niech dla każdej akcji ξ, $\mathbf{A}_\xi$ oznacza aksjomat przejść dla ξ. Jest to relacja między poprzednimi i nowymi wartościami, inaczej mówiąc – zbiór par wektorów stanu. Przykładowo, para $((0, 50, \gamma), (0, 25, \delta))$ należy do $\mathbf{A}_\gamma$, ponieważ możliwe jest wykonanie akcji γ przechodząc od wartości **x** = **0**, **y** = **50** i **pc** = γ do wartości **x** = **0**, **y** = **25** i **pc** = δ.

Niech $\mathbf{A}_1$ i $\mathbf{A}_2$ oznaczają zbiory akcji, odpowiednio, dla diagramu i programu. Formalnie, należy udowodnić następujące twierdzenie:

$\forall \zeta \in \mathbf{A}_2 \forall (\mathbf{v}, \mathbf{w}) \in \mathbf{A}_\zeta \exists \eta \in \mathbf{A}_1$, takie że $(\mathbf{F(v)}, \mathbf{F(v)}, \mathbf{F(w)}) \in \mathbf{A}_\eta$

Tę formułę odczytuje się następująco (wyrażenia w nawiasach kwadratowych odpowiadają poszczególnym częściom powyższego zapisu):
dla każdej akcji [ξ] z programu [z $\mathbf{A_2}$] i każdej zmiany wartości **x**, **y** i **pc** [z **v** na **w**] dopuszczalnej dla tej akcji [(**r**, **w**) z $\mathbf{A}_\xi$] istnieje akcja [η] z diagramu [z$\mathbf{A}_1$], taka że odpowiednia zmiana wartości **f** [z **F(v)** na **F(w)**] jest dopuszczalna dla tej akcji
[(**F(v)**, **F(w)**) należy do $\mathbf{A}_\eta$].

Co naprawdę zostało dowiedzione?

Formalnym znaczeniem specyfikacji według diagramu jest formuła:

$\exists$**f** taka, że $\mathbf{X}_1$

gdzie $\mathbf{X}_1$ jest formułą opisującą, jak może zmieniać się wartość **f**.
Znaczeniem programu jest formuła:
$\exists$ **x**, **y**, **pc** takie, że $\mathbf{X}_2$
gdzie $\mathbf{X}_2$ opisuje, jak mogą zmieniać się wartości **x**, **y** i **pc**.
Dowiedziono formuły
$(\exists \mathbf{x}, \mathbf{y}, \mathbf{pc}$ t.ż. $\mathbf{X}_2) \supset (\exists \mathbf{f}$ t. ż $\mathbf{X}_1)$

[1] Zwróćmy uwagę, że wydruk 2 nie jest programem; jest zwykłą matematyczną definicją **f** jako funkcji **x**, **y** i **pc**, zapisaną w notacji Dijkstry.

Tak więc, poprawność realizacji oznacza zwykłą, logiczną implikację – specyfikacja wynika z (specyfikacji) realizacji. Tę implikację udowodniono, wykazując że:

$$X_2 \supset \bar{X}_1$$

gdzie $\bar{X}_1$ jest formułą otrzymaną przez podstawienie za **f** w X_1 jej wartości wyrażonej przez **x**, **y** i **pc** według wydruku 2. Inaczej mówiąc, zakładając istnienie funkcji stanu **x**, **y** i **pc** spełniających X_2, dowiedziono istnienia funkcji **f** spełniającej X_1, jawnie konstruując wymaganą funkcję **f** w zależności od **x**, **y** i **pc**.

W pełnej specyfikacji, X_1 i X_2 powinny uwzględniać również dopuszczalne zmiany funkcji stanu sprzężenia. Jednakże, ponieważ twierdzenie nie zawiera kwantyfikacji dla funkcji stanu sprzężenia, takie uzasadnienie jest możliwe tylko wtedy, gdy X_1 i X_2 zawierają te same funkcje stanu sprzężenia, a zachowanie funkcji stanu sprzężenia zapewniane przez X_1 wynika z asercji o ich zachowaniu określonych w X_2. Jest to formalnym stwierdzeniem faktu, że sprzężenie musi być wyspecyfikowane (w X_1) na poziomie realizacji (przy użyciu tych samych funkcji stanu i akcji sprzężenia, co w X_2).

Dowód, że program jest poprawną realizacją diagramu można sprowadzić do implikacji, ponieważ obie specyfikacje są wyrażone za pomocą formuł tego samego systemu logicznego. To z kolei jest możliwe dlatego, że diagram przejść stanów interpretuje się przy użyciu pojęcia funkcji stanu **f**. Gdyby uznać, że diagram przedstawiony na rysunku specyfikuje zachowanie maszyny trójstanowej, a program – zachowanie maszyny 30-stanowej, wtedy odpowiednie formuły byłyby wyrażone w różnych systemach logicznych i nie byłoby jasne, co znaczy poprawność realizacji. Tradycyjnie definicje poprawności realizacji, oparte na odwzorowaniach zachowań pomijają zagadnienia dotyczące specyfikowania sprzężeń. Dokładniejsze omówienie tego problemu przedstawiono w [4].

BAZA DANYCH

Obecnie rozważę przykładową specyfikację mechanizmów sterowania współbieżnością dla bazy danych. Klienci bazy danych wykonują operacje przez wywoływanie procedury **exec** z argumentami wskazującymi rodzaj operacji. Są dwa rodzaje argumentów:
op – wskazujący modyfikację w bazie danych,
res – wskazujący udostępnienie wartości (zakładam, że procedury w Moduli 2 mogą udostępniać wartości).
Te argumenty są opisane formalnie jako funkcje. Choć procedura **exec** może być wywoływana współbieżnie przez wielu klientów, operacje są wykonywane tak, jakby występowały sekwencyjnie. Inaczej mówiąc, operacje bazy danych mają być wykonywane tak, jakby były atomowe [2].

```
type dbase  : ... ;
     value  : ... ;
     op_fcn : dbase → dbase;
     res_fcn : dbase → value;
internal state function data : dbase;
procedure exec(op: op_fcn; res: res_fcn) : value
  begin α: ⟨data_new = op(data_old) ∧ exec_new = res(data_old)⟩
  end
```

Wydruk 3. Specyfikacja bazy danych

Specyfikację przedstawiono na wydruku 3 używając notacji języka programowania. Stan bazy danych jest reprezentowany przez wewnętrzną funkcję stanu **data**. Sprzężenie wyspecyfikowano jako wywołanie procedury. Istnieje jedna akcja wewnętrzna α. W tej akcji **op** i **res** oznaczają argumenty wywołania procedury, a **exec** – wartość udostępnianą przez tę procedurę. Wywołanie procedury i udostępnienie przez nią wartości stanowią akcje sprzężenia, które nie są jawnie specyfikowane.

Jak interpretuje się wydruk 3 jako specyfikację metodą aksjomatu przejść?

[2] W języku baz danych atomowość oznacza, że operacja musi być całkowicie zakończona także w wypadku awarii. Jednakże w tym przykładzie nie rozważa się możliwości awarii.

Specyfikacja sprzężenia musi zawierać funkcje stanu, których wartości wskazują zbiór procesów aktualnie wykonujących procedurę **exec**, tzn. zbiór procesów, które wykonały wywołanie jeszcze nie zakończone udostępnieniem wartości. Muszą także istnieć funkcje stanu, które zawierają następujące informacje dla każdego z tych procesów:
– wartości **op** i **res**,
– komórkę sterowania programem wskazującą czy proces wykonuje wywołanie, czy jest w punkcie α, lub czy pobiera udostępnianą wartość,
– wartość **exec**, która musi być udostępniona, gdy proces wykonał akcję α.

Ze specyfikacji na wydruku 3 wynika, że cała operacja w bazie danych musi być wykonana jako jedna atomowa akcja. Czy to nie wyklucza współbieżności?

Specyfikacja zakłada istnienie funkcji stanu **data**, która zmienia się atomowo, jednak nie zakłada, że zmiany w bazie danych muszą być w rzeczywistości wykonywane atomowo. Specyfikacja z wydruku 1 jest realizacją specyfikacji z rysunku w pierwszej części tego artykułu, mimo że operacje wrzucenia monety półdolarowej i wydania puszki z wodą sodową składają się z dwóch akcji atomowych na rysunku i sześciu akcji atomowych na wydruku. Poprawności realizacji dowiedziono definiując funkcję **f** w taki sposób, że tylko dwie spośród sześciu akcji zmieniają **f** w sposób dozwolony na rysunku, a pozostałe cztery akcje pozostawiają ją bez zmian.

W ten sam sposób, zmiana w bazie danych reprezentowana przez wykonanie pojedyńczej akcji atomowej α z wydruku 3 może być zrealizowana jako ciąg tysięcy akcji atomowych. Poprawność realizacji oznacza, że funkcję stanu **data** można zdefiniować jako funkcję uzależnioną od funkcji stanu realizacji w taki sposób, że tylko jedna z tych tysięcy akcji atomowych zmienia jej wartość zgodnie z wydrukiem 3.

Funkcja stanu musi być tak zdefiniowana, że pojedyńcza akcja atomowa spowoduje nagłe wykonanie całej, choćby bardzo skomplikowanej, operacji w bazie danych, mimo że każda akcja atomowa dokonuje jedynie niewielkiej zmiany w tej bazie. Jak to jest możliwe?

Jednym sposobem wyjaśnienia, jak to się odbywa, jest podanie przykładu. Takim przykładem jest specyfikacja i realizacja kolejki FIFO przedstawiona w [3]. Specyfikacja zakłada, że operacje dodania elementu do kolejki i usunięcia elementu z kolejki są atomowe, lecz realizacja, która dodaje i usuwa elementy jednocześnie jest uznana za poprawną.

Metoda dowodu jest uogólnieniem asercyjnych metod dowodzenia właściwości bezpieczeństwa programów współbieżnych [5]. W metodach asercyjnych konstruuje się niezmiennik, będący boolowską funkcją stanu o nigdy nie zmieniającej się wartości. W metodzie aksjomatu przejść tworzy się natomiast funkcje stanu, których wartości zmieniają się tylko w sposób opisany przez aksjomaty przejść.

INNA SPECYFIKACJA BAZY DANYCH

Mechanizm kontroli współbieżności wyspecyfikowany na wydruku 3 nazywa się serializacją [2], ponieważ operacje w bazie danych są wykonywane tak, jakby występowały w pewnym porządku seryjnym. Jednakże wydruk nie specyfikuje w sposób najbardziej ogólny serializacji, ponieważ założono, że odczyt i zapis bazy danych występuje między wywołaniem procedury **exec** a kolejnym udostępnieniem wartości. Wynika stąd, że jeśli jedno wywołanie procedury **exec** spowoduje udostępnienie wartości zanim nastąpi inne wywołanie, to operacja wykonana wskutek pierwszego wywołania poprzedza w porządku seryjnym operację wykonaną wskutek drugiego wywołania. Niektóre algorytmy kontroli współbieżności, uważane za podatne do serializacji, nie mają tej właściwości. Tak więc, obecnie rozważę ogólniejszą specyfikację serializacji bazy danych przedstawioną na wydruku 4.

Sprzężenie nowej specyfikacji jest takie samo jak poprzedniej, tzn. – procedura **exec** o dwóch argumentach określających

operację. Jednakże, zamiast natychmiastowego wykonania operacji akcja β powoduje wybranie do udostępnienia całkowicie dowolnej wartości **V** i zachowanie jej razem z argumentami wywołania procedury w multizbiorze (ang. bag) **saved_ops**, zawierającym operacje do późniejszego wykonania[3]. Oddzielna, asynchroniczna akcja γ służy do wykonania tych operacji w jakiś czas później. Klauzula **internal process** oznacza, że odpowiednie akcje są wykonywane niezależnie od akcji wykonywanych wskutek wywołania procedury **exec**. Równość:

$$\mathbf{v} = \mathbf{r}(\mathbf{gdata}_{old})$$

w specyfikacji akcji γ oznacza, że akcja jest wykonywana tylko wtedy, gdy stan bazy danych jest taki, że wynik który już został wybrany (przez wykonanie akcji β polegającej na umieszczeniu trójki **(o, r, v)** w multizbiorze **saved_ops**) był poprawny.

Ponieważ jest to specyfikacja bezpieczeństwa, nie zakłada, że akcja γ będzie kiedykolwiek wykonywana. Zakłada jedynie, że akcja γ nie może być operacją w bazie danych, o ile jej wynik nie zgadza się z wynikiem, który już został udostępniony przez akcję β. Należy również założyć właściwość żywotności, polegającą na tym że każda operacja zachowana w multizbiorze **saved_ops** będzie w końcu wykonana. Specyfikacja tej właściwości zostanie przedstawiona w następnej części artykułu.

```
type dbase : ... ;
     value : ... ;
     op_fcn : dbase → dbase;
     res_fcn : dbase → value;
internal state function
   gdata      : dbase;
   saved_ops  : bag of (op_fcn, res_fcn, value);
procedure exec(op: op_fcn; rs: res_fcn) : value
 begin β: (∃r s.t. exec_new = r ∧
           saved_ops_new = saved_ops_old ∪ {(op, rs, r)})
 end
internal process
 begin loop γ: (∃(o, r, r) ∈ saved_ops_old s.t.
                gdata_new = o(gdata_old) ∧
                r = r(gdata_old) ∧
                saved_ops_new = saved_ops_old − {(o, r, r)})
       endloop
 end
```

Wydruk 4. Ogólniejsza specyfikacja bazy danych

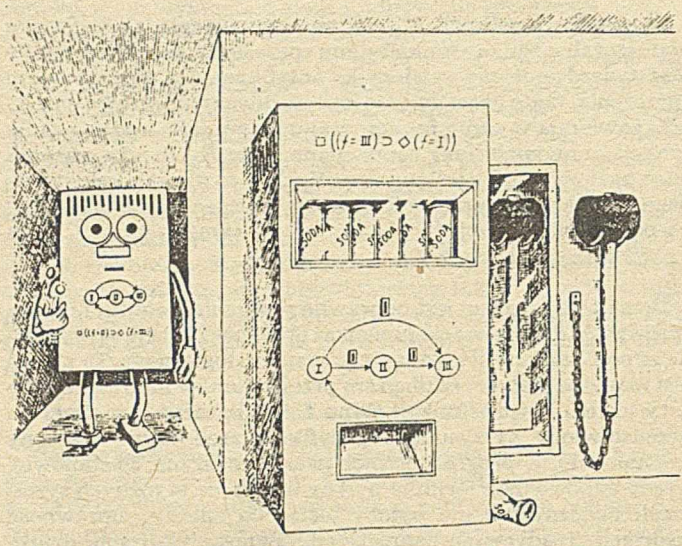
Ta specyfikacja jest całkowicie niejasna. Wymaga ona, aby procedura **exec** *zgadła, jaki będzie poprawny wynik operacji w bazie danych przed jej faktycznym wykonaniem. Jak można zrealizować taką specyfikację?*

Wydruk 4 jest niejasny tylko wtedy, gdy traktuje się go jako opis, jak ma być zrealizowana procedura **exec**. Jednakże specyfikacja opisuje tylko, co ma być zrobione, natomiast program opisuje, jak to ma być zrobione. Wydruk 4 opisuje obserwowalne zachowanie bazy danych, nie zawiera natomiast żadnej formalnej asercji, jak takie zachowanie ma być zrelizowane.

Jest istotne, aby sobie uświadomić, że choć specyfikacja metodą aksjomatu przejść może wyglądać jak program, pozornie opisujący realizację systemu, to w rzeczywistości specyfikuje ona tylko zewnętrznie widoczne zachowanie, tj. możliwe zmiany funkcji stanu sprzężenia. Wewnętrzne funkcje stanu, jak **saved_ops**, nie muszą występować w żadnej jawnej postaci w strukturze danych realizacji. W rzeczywistości, jak wyjaśniam poniżej, nie muszą one występować w ogóle.

Jeśli specyfikacja na wydruku 4 jest ogólniejsza od specyfikacji na wydruku 3, to ta druga powinna być realizacją pierwszej. Jednakże, realizacja wykonuje mniej akcji niż specyfikacja, tj. pojedyncza akcja α odpowiada operacji w bazie danych określonej w specyfikacji przez akcję β (umieszczenia operacji w multizbiorze **saved_ops***) i akcję γ (zmiany w bazie danych). Jak można udowodnić, że jest to poprawna realizacja?*

Podobnie jak w dowodzie dla przykładu dotyczącego automatu z wodą sodową należy zdefiniować funkcje stanu specyfikacji, **gdata** i **saved_ops**, w zależności od stanu realizacji – **data**. Niech więc funkcja **gdata** będzie równa funkcji **data**, a **saved_ops** zawsze równa się pustemu **multizbiorowi**. Ponownie należy wykazać, że każda akcja realizacji zmienia wartości funkcji stanu specyfikacji. Jednakże, pomijam wymaganie aby każda akcja realizacji odpowiadała co najwyżej pojedynczej akcji specyfikacji i dopuszczam, aby odpowiadała ciągowi takich akcji. Na wydruku 3 jest tylko jedna akcja wewnętrzna α. Wykonanie akcji α powoduje takie same zmiany funkcji **gdata** i **saved_ops** jak wykonanie akcji β z następującym po niej wykonaniem akcji γ. Akcja γ powoduje natychmiastowe wykonanie operacji umieszczonej w multizbiorze **saved_ops** wskutek akcji β, co odpowiada w efekcie opróżnieniu multizbioru **saved_ops** i dokonaniu żądanej zmiany funkcji **gdata**.



Czy zawsze dopuszczalne jest realizowanie ciągu akcji specyfikacji za pomocą pojedynczej akcji realizacji?

Do tego, aby ciąg akcji specyfikacji był realizowany jako pojedyncza akcja realizacji, muszą być spełnione następujące dwa warunki:

- co najwyżej jedna akcja specyfikacji może być akcją sprzężenia,
- wszystkie akcje razem wzięte (tj. akcje specyfikacji i realizacji) muszą być wykonywane przez ten sam człon, tj. albo przez system albo przez otoczenie.

Te dwa warunki wykluczają realizacje patologiczne.

Przypadek realizowania kilku akcji specyfikacji za pomocą pojedynczej akcji występuje tylko wtedy, gdy trzeba wykazać, że jedna specyfikacja jest co najmniej tak samo ogólna jak inna. Tak na przykład wykazałem, że specyfikacja z wydruku 4 jest co najmniej równie ogólna jak specyfikacja z wydruku 3. W rzeczywistych realizacjach, pojedyncza akcja specyfikacji jest zwykle realizowana za pomocą dziesiątek, a niekiedy nawet tysięcy oddzielnych akcji.

Jakie jest formalne uzasadnienie poprawności realizowania kilku akcji specyfikacji za pomocą pojedynczej akcji?

Jak pamiętamy, formuła reprezentująca specyfikację w metodzie aksjomatu przejść ma postać

$\exists\, f_1,...,f_n$ takie, że X

gdzie $\mathbf{f_1}, \mathbf{f_n}$, są wewnętrznymi funkcjami stanu. Powód, dla którego dopuszcza się implementowanie kilku akcji za pomocą jednej, wynika z formalnego znaczenia kwantyfikacji egzystencjalnej dla funkcji stanu, co przedstawiono w [1]. W dowodzie, że z formuły reprezentowanej przez realizację wynika formuła reprezentowana przez specyfikację, kwantyfikacja egzystencjalna nad wewnętrznymi funkcjami stanu umożliwia stwierdzenie, że wykonanie polegające na rozdzieleniu jednej akcji na kilka kolejnych akcji jest równoważne pierwotnemu wykonaniu, jeśli dodatkowe akcje zmieniają tylko wewnętrzne funkcje stanu. Jednakże, wyjaśnienie, dlaczego tak jest, wymaga skorzystania z kilku subtelności logiki temporalnej, co wykracza poza zakres tego artykułu.

dokończenie na s. 14

[3] W oryginale Autor używa terminów **bag** i **multiset**, dla których nie znalazłem dobrego odpowiednika polskiego. Wyjaśnia przy tym w przypisie, że chodzi o zbiór, w którym ten sam element może wystąpić więcej niż raz (przyp. tłum.).

Od pewnego czasu społeczność związana z bazami danych objawia silne zainteresowanie problematyką zintegrowania technologii baz danych z innymi technikami stosowanymi w informatyce. Integracja ta zmierza głównie w kierunkach skojarzenia technologii baz danych z technologią programowania w języku logiki oraz z technologią sztucznej inteligencji. W pierwszym wypadku celem jest zwiększenie mocy sprzężenia z systemem bazy danych do poziomu, który zapewniają uniwersalne języki programowania. Efektem pracy w tej dziedzinie są szczególnie propozycje języków obiektowych, języków funkcjonalnych i imperatywnych oraz – opartych na logice.

Drugim ważnym kierunkiem rozwoju technologii baz danych jest dążenie do skojarzenia jej z technologią sztucznej inteligencji. W zamierzeniu ma to doprowadzić do takiego rozwinięcia systemów baz danych, aby uzyskały one właściwości funkcjonalne charakterystyczne dla systemów ekspertowych. Celem integracji jest więc tworzenie systemów, nie jak dotychczas z bazą danych, lecz z bazą wiedzy. W szczególności, jednym z aspektów integracji technologii baz danych z technologią sztucznej inteligencji jest włączenie do systemów baz danych osiągnięć z dziedziny przetwarzania języka naturalnego, czego rezultatem mogłaby być możliwość komunikowania się z bazą danych w języku naturalnym. Jest to niewątpliwie perspektywa interesująca i bardzo atrakcyjna z punktu widzenia praktyki. Czy jednak i na ile jest to perspektywa realna? Tę i inne kwestie dotyczące problemów przetwarzania języka naturalnego pozwoli, jak sądzimy, przybliżyć poniższy artykuł.

WIKTOR RZECZKOWSKI

AGNIESZKA MYKOWIECKA
ANNA OSTASZEWSKA
Instytut Podstaw Informatyki
Polska Akademia Nauk
Warszawa

Rozmowa z komputerem

Problematyka komunikacji z komputerem w języku naturalnym jest nieomal białą plamą na mapie polskiej informatyki. Prace z tej dziedziny prowadzone w naszym kraju są nieliczne i mają, jak dotąd, wyłącznie eksperymentalny charakter. Jesteśmy dopiero na samym początku drogi, podczas gdy już teraz na świecie są dostępne handlowo systemy posługujące się językiem naturalnym (a dokładniej – pewnym podzbiorem języka naturalnego) [5]. Sądzimy, że dziedzina przetwarzania języka naturalnego jest niezmiernie ważna dla rozwoju nie tylko informatyki, lecz także całej gospodarki i z tego powodu warta jest szerszej popularyzacji. W niniejszym artykule przedstawiamy ogólne wprowadzenie w tę tematykę (patrz też [3]).

CO TO ZNACZY „ROZUMIEĆ"?

Komputer potrafi w krótkim czasie wykonać obliczenia, nad którymi człowiek musiałby pracować nieraz wiele lat. Fakt ten na wielu osobach do dziś robi duże wrażenie. Ale cóż to za osiągnięcie – zrozumieć lub wypowiedzieć proste zdanie w języku naturalnym? Przecież to potrafi każde dziecko. Okazuje się jednak, że jest to problem niezwykle złożony. Wiele aspektów tego zagadnienia przedstawia w swoich publikacjach R. Schank (np. [9]). Nie znamy mechanizmów rządzących procesem rozumienia wypowiedzi przez człowieka. Owszem, istnieją różne (bardziej lub mniej kompletne) hipotezy na ten temat, ale czy kiedykolwiek będziemy mieli pewność, że któraś z nich jest prawdziwa?

W badaniach w dziedzinie sztucznej inteligencji zaznaczyły się dwa kierunki. Pierwszy z nich to dążenie do jak najlepszego poznania procesów myślowych u człowieka, aby później budować systemy stosujące takie same metody, jakie stosuje człowiek. To podejście wymaga długotrwałych badań podstawowych przy współudziale przedstawicieli takich dyscyplin naukowych, jak psychologia, lingwistyka czy fizjologia [6]. Przedstawiciele drugiego kierunku twierdzą, że należy dążyć do uzyskania wymaganych rezultatów (tzn. do stworzenia systemów zachowujących się podobnie jak człowiek), natomiast metoda, za pomocą której te rezultaty zostaną osiągnięte, jest obojętna.

Niezależnie od stosowanego podejścia należy wiedzieć, czego oczekuje się od systemu przetwarzającego język naturalny. Potocznie mówi się, że maszyna powinna rozumieć wypowiedzi w języku naturalnym. Okazuje się, że nie tak łatwo jest uściślić znaczenie słowa „rozumieć". Czy system ELIZA, który na wypowiedź człowieka:

„Mój ojciec jest wyjątkowo despotyczny"

odpowie:

Mgr AGNIESZKA MYKOWIECKA ukończyła w 1982 r. Wydział Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego. Obecnie pracuje w Instytucie Podstaw Informatyki PAN w Zespole Komunikacji Człowiek-Komputer. Zajmuje się problemami związanymi z przetwarzaniem języka naturalnego, a szczególnie generacją tekstów.

Mgr ANNA OSTASZEWSKA ukończyła w 1986 r. Wydział Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego. Obecnie pracuje w Instytucie Podstaw Informatyki PAN w Zespole Komunikacji Człowiek-Komputer. Zajmuje się zagadnieniami reprezentacji wiedzy w systemach z komunikacją w języku naturalnym.

„POWIEDZ MI COŚ WIĘCEJ O SWOJEJ RODZINIE"
rozumie wprowadzone zdanie? Działanie tego systemu (stworzonego w 1966 roku przez Weizenbauma) polega na wyszukiwaniu w podanym na wejściu zdaniu wcześniej zdefiniowanych słów kluczowych. Odpowiedź systemu jest wybierana spośród schematów zdaniowych przyporządkowanych poszczególnym słowom kluczowym. Dla dużego zbioru rozpoznawanych słów kluczowych i bogatego repertuaru odpowiadających im schematów działanie systemu stwarza wszelkie pozory rozumienia wypowiedzi użytkownika.

Dość trudno jest podać kryterium pozwalające rozstrzygnąć, czy dany system rzeczywiście rozumie wypowiedź człowieka. Na ogół, za takie kryterium można przyjąć warunek mówiący, że reakcja systemu na każdą wypowiedź użytkownika powinna mieć sens, a w szczególności, że system powinien dokładnie wykonać polecenie wyrażone w tej wypowiedzi. Już przy tak prostym kryterium trzeba zdyskwalifikować taki system, jak ELIZA. Jeśli bowiem ELIZA nie odnajdzie w wypowiedzi użytkownika żadnego ze znanych słów kluczowych, to będzie ratować się użyciem wypowiedzi możliwie neutralnych, co jednak nie zawsze ma sens.

Dlaczego tak trudno jest osiągnąć rozumienie wypowiedzi przez system komputerowy? Zauważmy, że nawet dla zrozumienia prostych, pojedynczych zdań konieczna jest znajomość:
– poszczególnych słów występujących w zdaniu,
– form gramatycznych poszczególnych wyrazów,
– składni języka, czyli reguł sterujących budową zdań.
Oprócz tego niezbędna jest, często trudna do sprecyzowania, ogólna wiedza o świecie (lub o pewnym wycinku rzeczywistości, którego dotyczy konwersacja). To ta właśnie wiedza o świecie sprawia, że wypowiedź:
„Mimo deszczu i wiatru poszliśmy na spacer"
jest dla nas zrozumiała i naturalna, natomiast podobnie zbudowane zdanie:
„Mimo pięknej pogody poszliśmy na spacer"
wydaje nam się dość dziwne. Nasza ocena sensowności wypowiedzi wynika tu z różnych informacji dotyczących pogody, ludzkiego zachowania itd. Tylko wyposażenie komputera w tego rodzaju ogólną wiedzę o świecie pozwoliłoby na poprawną i głęboką analizę semantyczną zdań języka naturalnego. Niestety, jest to zadanie niezwykle trudne i złożone (wrócimy do tego zagadnienia nieco dalej).

CO OZNACZA KOMUNIKACJA W JĘZYKU NATURALNYM?

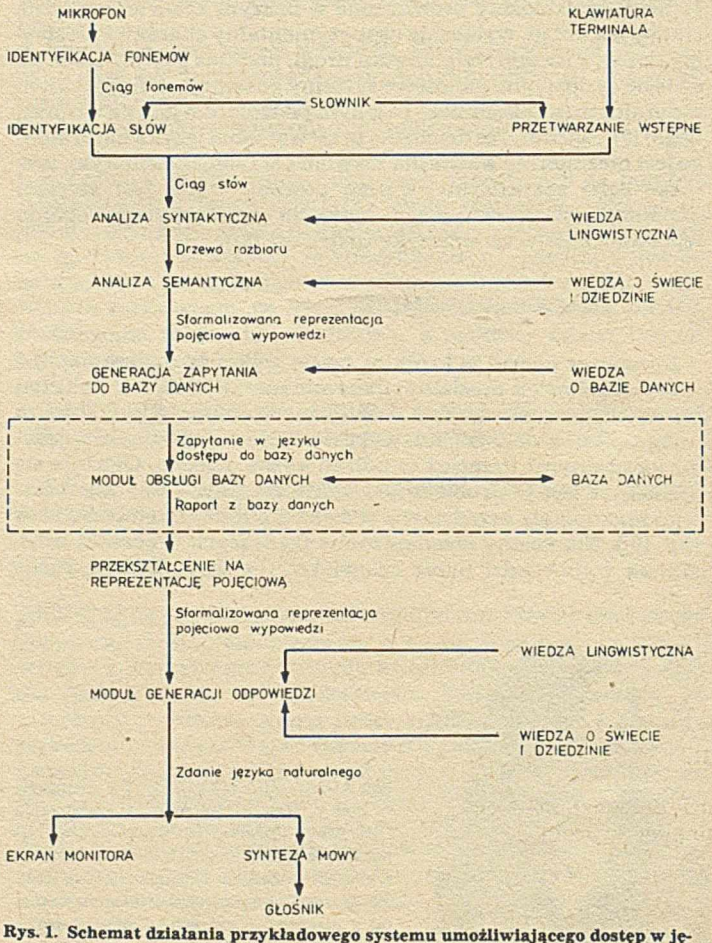
Wyobraźmy sobie, że w rejestracji naszej przychodni rejonowej zainstalowano komputer. Oto, jak mogłaby wyglądać konwersacja komputera z przybyłym do przychodni pacjentem (wypowiedzi pacjenta wydrukowano kursywą).
Dzień dobry
DZIEŃ DOBRY, PODAJ MI SWOJE IMIĘ I NAZWISKO.
Anna Ostaszewska
JAKI JEST TWÓJ ADRES?
Marymoncka 34 m 54
DO JAKIEGO SPECJALISTY CHCESZ SIĘ UDAĆ?
do internisty
PRZYKRO MI, ALE TWÓJ LEKARZ JEST NA URLOPIE. CZY CHCESZ BYĆ PRZYJĘTA PRZEZ INNEGO LEKARZA?
tak
PRZESYŁAM TWOJĄ KARTĘ DO DR ABACKIEGO, GABINET 36, I PIĘTRO.
dziękuję
Prawdopodobnie kilkunastoletni uczeń programujący w Basicu na swoim Spectrum czy Commodore potrafiłby w niedługim czasie napisać program, obsługujący tego rodzaju rozmowy. Program ten nie miałby jednak nic wspólnego z przetwarzaniem języka naturalnego. Kolejność zadawania pytań jest tu ściśle ustalona, a pytania mogą być przechowywane w postaci stałych tekstów. Dla każdego pytania należałoby określić skończony zbiór akceptowalnych odpowiedzi, a wprowadzona przez pacjenta odpowiedź sterowałaby dalszym przebiegiem konwersacji. Program musiałby ponadto dysponować niewielką bazą danych zawierającą informacje o pacjentach przychodni i o przyjmujących lekarzach. Tego rodzaju program jest jedynie przyjemną dla użytkownika modyfikacją systemu menu (dla potrzeb przychodni mógłby on być zupełnie wystarczający).

Podobnie jak opisany powyżej system, nie mają nic wspólnego z przetwarzaniem języka naturalnego systemy, nazywane w języku angielskim *single word systems*, które akceptują jedynie ograniczony zbiór prostych poleceń. Przykładem takiego systemu może być program sterujący urządzeniami klimatyzacyjnymi, który akceptuje pojedyńcze rozkazy w rodzaju: *zwiększ wilgotność, obniż temperaturę* itp.

Dotąd zajmowaliśmy się rozważaniami na temat tego, co na pewno nie jest komunikacją w języku naturalnym. Obecnie spróbujemy określić, kiedy taka komunikacja występuje. System przetwarzający język naturalny umożliwia użytkownikom swobodne wprowadzanie dowolnych zdań języka naturalnego, a odpowiedzi maszyny również są sformułowane w postaci pełnych zdań tego języka. W idealnym przypadku użytkownik powinien odnosić wrażenie, że rozmawia za pośrednictwem terminala z innym człowiekiem.

Dla osiągnięcia tego celu tworzone są coraz doskonalsze programy zajmujące się analizą i syntezą wypowiedzi. Prowadzone są badania zarówno nad przetwarzaniem języka pisanego, jak i nad analizą sygnału mowy oraz syntezą mowy. Na rys. 1 przedstawiono przykładowy schemat działania systemu, który umożliwia dostęp w języku naturalnym do bazy danych. System taki mógłby powstać z tradycyjnego systemu obsługi bazy danych (z dostępem realizowanym za pośrednictwem sformalizowanego języka zapytań) przez obudowanie go modułami obsługującymi konwersację w języku naturalnym. Na schemacie z rys. 1 ta najbardziej „pierwotna" część systemu jest obwiedziona linią przerywaną.

Na rysunku przedstawiono podstawowe moduły programowe wchodzące w skład prawie każdego systemu przetwarzającego język naturalny. Rola i sposób działania większości z tych modułów są omówione poniżej.



Rys. 1. Schemat działania przykładowego systemu umożliwiającego dostęp w języku naturalnym do bazy danych

Słownik

System przetwarzający język naturalny musi dysponować obszernym słownikiem. Każdy element ciągu wyrazów podanego na wejściu jest odszukiwany w słowniku, skąd pobiera się opis cech gramatycznych wyrazu (jaką jest częścią mowy, w jakim przypadku, liczbie, osobie itd.). Niektóre systemy stosują metodę pomijania w trakcie analizy wyrazów, których nie odnaleziono w słowniku – wówczas znaczenie wypowiedzi jest ustalane na podstawie pozostałych, znanych systemowi słów. Może to jednak prowadzić do nieporozumień. Toteż w wypadku zastosowań wymagających dużej niezawodności systemu zdania zawierające nieznane słowa nie są akceptowane, a użytkownik jest proszony o przeformułowanie wypowiedzi.

Inaczej wygląda kwestia utworzenia słownika w wypadku języków o ubogiej fleksji (takich jak język angielski), a inaczej, gdy mamy do czynienia z językiem o tak rozbudowanej i nieregularnej fleksji, jak język polski [8]. Przykładowo, dla każdego polskiego przymiotnika należałoby przechowywać aż 126 jego form (3 rodzaje × 3 stopnie × 2 liczby × 7 przypadków). Oczywiście dla poszczególnych części mowy obowiązują pewne reguły odmiany, ale – niestety – jest ich bardzo dużo, a ponadto występuje wiele wyjątków. Toteż może się okazać, że dla języków fleksyjnych przechowywanie w słowniku wszystkich możliwych form danego słowa (mimo iż niezmiernie pamięciochłonne) jest bardziej opłacalne, niż próby automatyzacji odmiany.

Widzimy już, że pojemność słownika jest pierwszą barierą ograniczającą podzbiór języka, którym będzie umiał posługiwać się komputer. Dla sprawnego funkcjonowania systemu potrzebne są pamięci o ogromnej pojemności i bardzo krótkim czasie dostępu. Zawsze jednak zbiór słów zawartych w leksykonie będzie skończony, tak więc zawsze będzie mogło zdarzyć się, że człowiek użyje nieznanego komputerowi słowa.

Proces analizy

Zajmiemy się teraz bliżej procesami analizy syntaktycznej i semantycznej wypowiedzi wprowadzonej przez użytkownika. Zagadnienia automatycznej analizy składniowej są najlepiej poznane spośród wszystkich problemów związanych z przetwarzaniem języka naturalnego [12]. Tutaj omówione zostanie działanie analizatora w systemie o architekturze przedstawionej na rys. 1.

Analizator syntaktyczny otrzymuje na wejściu ciąg słów, przy czym każde z nich ma już swój opis, zaczerpnięty ze słownika. Zauważmy, że niektóre wyrazy mogą mieć kilka takich opisów. Na przykład, wyraz *nerwowi* może być zarówno celownikiem liczby pojedyńczej rzeczownika *nerw*, jak i mianownikiem rodzaju męskiego liczby mnogiej przymiotnika *nerwowy*.

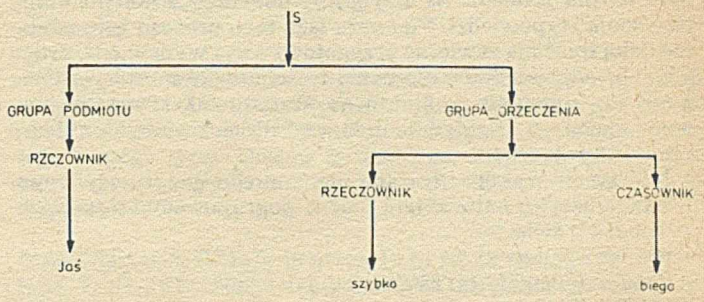
Analizator syntaktyczny (ang. *parser*) opiera się w działaniu na sformalizowanym opisie składni przetwarzanego języka (jedyny istniejący formalny opis składni języka polskiego znajduje się w pracy S. Szpakowicza [10]. Działanie takiego analizatora rozpatrzymy na przykładzie następującej prostej gramatyki bezkontekstowej:

S → GRUPA_PODMIOTU GRUPA_ORZECZENIA
GRUPA_PODMIOTU → RZECZOWNIK | PRZYMIOTNIK RZECZOWNIK
RZECZOWNIK → pies | Jaś | kot
PRZYMIOTNIK → mały | miły
GRUPA_ORZECZENIA → CZASOWNIK | PRZYSŁÓWEK CZASOWNIK
CZASOWNIK → biega | skacze
PRZYSŁÓWEK →szybko | wesoło

gdzie **S** jest symbolem początkowym gramatyki, symbole nieterminalne wydrukowane są dużymi literami, a terminalne małymi.

Jeśli teraz na wejściu pojawi się zdanie: *„Jaś szybko biega"*, to analizator syntaktyczny utworzy drzewo rozbioru przedstawione na rys. 2.

W procesie analizy syntaktycznej wypowiedzi należy również uwzględnić takie reguły, jak zgodność podmiotu i orzeczenia pod względem liczby, rodzaju, osoby itd. Zdania, dla których nie istnieje rozbiór syntaktyczny, są najczęściej odrzucane jako niepoprawne (choć w zasadzie są one jedynie niezgodne z zaimplementowaną składnią). Dla niektórych zdań parser generuje więcej niż jedno drzewo rozbioru. Przykładowo, w zdaniu *„Lekarstwo podajemy dziecku w postaci lewatywy"* fraza *w postaci lewatywy* może być związana zarówno z rzeczownikiem *dziecko*, jak i *lekarstwo*. Dopiero analiza semantyczna może rozstrzygnąć tę dwuznaczność, przy czym analizator semantyczny musi umieć stwierdzić, że dziecko nie może być w postaci lewatywy, natomiast lekarstwo – jak najbardziej. O ile więc parser posługuje się wyłącznie wiedzą lingwistyczną, czyli sformalizowanym opisem budowy zdań danego języka, o tyle analizator semantyczny musi dysponować wiedzą o znaczeniu przetwarzanych słów i fraz. Wiedza ta może niekiedy nie wystarczać do rozstrzygnięcia dwuznaczności – wówczas system może zadać użytkownikowi pytanie uściślające lub odrzucić wypowiedź. Może się też zdarzyć, że dla danej wypowiedzi żaden z rozbiorów syntaktycznych nie jest poprawny pod względem semantyki i taką wypowiedź odrzuca się jako pozbawioną sensu.



Rys. 2. Drzewo rozbioru dla zdania „Jaś szybko biega"

Zadaniem analizatora semantycznego jest wygenerowanie sformalizowanej reprezentacji wypowiedzi wprowadzonej przez użytkownika. Istnieje wiele schematów reprezentacji, spośród których najczęściej stosowane są sieci semantyczne, ramy i reprezentacje logiczne. Bez względu na to, który schemat reprezentacji zostanie wybrany, musi on umożliwiać wyrażenie całej istotnej treści przetwarzanego zdania.

Proces analizy kończy się pomyślnie, jeśli otrzymamy jednoznaczną formalną reprezentację wypowiedzi.

Przypomnijmy, jakie problemy napotyka się w czasie analizy:
– niejednoznaczny opis leksykalny wyrazu lub brak wyrazu w słowniku,
– więcej niż jeden rozbiór gramatyczny zdania lub brak poprawnego rozbioru,
– niejednoznaczna interpretacja semantyczna lub niepoprawność semantyczna wypowiedzi.

Jeśli uda się szczęśliwie pokonać wszystkie te trudności, to etap analizy można uważać za zakończony. Dalsze przetwarzanie formalnej reprezentacji wypowiedzi zależy od przeznaczenia systemu. W wypadku systemu o architekturze przedstawionej na rys. 1 reprezentacja ta jest przekształcana na zapytanie w języku dostępu do bazy danych.

Generowanie odpowiedzi

Zadanie generowania wypowiedzi w języku naturalnym można rozpatrywać jako odwrotność procesu analizy. Na wejściu modułu generowania odpowiedzi wprowadzana jest sformalizowana reprezentacja znaczenia zdania, na wyjściu zaś otrzymywane jest zdanie w języku naturalnym, wyrażające zadaną treść.

Prace związane z zagadnieniem generowania zdań w języku naturalnym podjęto o wiele później niż badania nad analizą języka naturalnego. Wynikało to przede wszystkim z faktu, że dopiero po osiągnięciu pewnych rezultatów w zakresie opracowania formalnego zapisu znaczenia wypowiedzi można było

przystąpić do automatyzacji przekształcania zapisu formalnego na tekst w języku naturalnym. Mimo intensywnych badań w tym kierunku nie uzyskano jeszcze zadowalających całościowych rozwiązań. Budowane obecnie systemy generujące wypowiedzi są bardzo silnie zdeterminowane przez rodzaj zastosowania, dla jakiego powstają [7].

Najprostszym zadaniem realizowanym przez moduły generowania tekstu jest synteza pojedynczego zdania na podstawie formalnego zapisu jego wymaganego znaczenia. Znacznie bardziej skomplikowane jest tworzenie dłuższych fragmentów tekstu lub prowadzenia dialogu. W wypadku formułowania dłuższych wypowiedzi wymagane jest staranne planowanie kolejności przedstawiania poszczególnych faktów oraz uwzględnianie powiązań pomiędzy następującymi po sobie zdaniami. Ponadto, konieczna jest różnorodność stylu pozwalająca na tworzenie tekstów łatwo czytelnych, nie nużących jednostajnością sformułowań. Prowadzenie dialogu wymaga dodatkowo pamiętania przebiegu rozmowy, umiejętności tworzenia i rozpoznawania odwołań do obiektów i faktów wcześniej wspomnianych oraz wyboru formy najbardziej adekwatnej w danym momencie dialogu.

Problemy związane z generowaniem tekstu w języku naturalnym można podzielić na trzy grupy obejmujące kolejne etapy tworzenia wypowiedzi. Pierwszą fazą tego procesu jest ustalenie, jaką treść ma zawierać przygotowywana wypowiedź, drugą – wybór odpowiednich dla danej treści środków retorycznych, a trzecią – nadanie ostatecznego kształtu tak sformułowanej wypowiedzi – z uwzględnieniem wszystkich wymogów składni i morfologii języka.

Informacje wykorzystywane przez moduł generujący wypowiedzi w języku naturalnym można pogrupować następująco:
- wiedza o świecie:
  - wiedza ogólna,
  - wiedza szczegółowa, zależna od dziedziny;
- wiedza lingwistyczna:
  - zasady wyboru poszczególnych konstrukcji językowych w zależności od sytuacji i kontekstu językowego,
  - zasady organizowania zdań w dłuższe wypowiedzi,
  - reguły budowy zdań, tworzenia faz, uzgadniania cech leksykalnych,
  - słownictwo, zasady morfologii.

ZASTOSOWANIA PRZETWARZANIA JĘZYKA NATURALNEGO

Systemy przetwarzające język naturalny możemy podzielić na dwie kategorie:
- systemy ułatwiające korzystanie z innych programów,
- systemy służące do wykonywania różnorodnych operacji na tekstach w języku naturalnym.

Powstawanie systemów pierwszego rodzaju wiąże się z coraz powszechniejszym użyciem komputerów w różnych dziedzinach gospodarki i życia codziennego. Sposób komunikowania się z obecnie dostępnymi programami użytkowymi jest dla wielu osób (nie mających przygotowania informatycznego) nieczytelny i skomplikowany ze względu na jego sformalizowaną postać. Wykorzystanie w codziennej pracy programów komputerowych wymaga włożenia niemałego wysiłku w nauczenie się zasad zadawania pytań i rozumienia odpowiedzi udzielanych przez program. Możliwość komunikowania się z komputerem w języku naturalnym stanowi więc ogromne ułatwienie dla użytkownika.

Wśród systemów drugiego typu mogą znajdować się następujące ich rodzaje:
- inteligentne edytory tekstów, sprawdzające poprawność ortograficzną, składniową i być może semantyczną wprowadzanych zdań,
- programy tłumaczące teksty z jednego języka naturalnego na inny,
- programy służące do nauki języka,
- programy tworzące opisy słowne sytuacji, o których mamy informacje w postaci formalnej (np. opisywanie zdjęć),
- programy tworzące bazy danych lub bazy wiedzy na podstawie informacji tekstowej,
- programy służące do badań lingwistycznych.

Powyższa lista daje pewne wyobrażenie o szerokich możliwościach zastosowania systemów przetwarzających język naturalny, choć z pewnością nie wyczerpuje wszystkich możliwości tych zastosowań. Kilka interesujących przykładów zastosowań przetwarzania języka naturalnego znaleźć można w publikacjach [1,4].

* * *

W artykule omówiono w sposób bardzo ogólny podstawowe problemy związane z próbą umożliwienia komunikacji człowieka z systemami komputerowymi w języku naturalnym. Prace nad tym zagadnieniem są prowadzone od wielu lat w licznych informatycznych ośrodkach badawczych. Osiągnięte do tej pory wyniki są tylko częściowe i pozwalają na realizację systemów o bardzo ograniczonych możliwościach ich zastosowania. Niektóre zagadnienia zostały rozwiązane zadowalająco, np. istnieje wiele formalizmów opisujących w sposób dość dobry składnię języka. Opracowano też i zaimplementowano wiele algorytmów analizy syntaktycznej wypowiedzi [2,11]. Inne problemy są nadal przedmiotem wielu prób. Do tych zagadnień należy na przykład opis semantyki wypowiedzi.

Osobom szczególnie zainteresowanym tematyką przetwarzania języka naturalnego polecamy poniższe publikacje. Zagadnienia te są omawiane również w każdej przeglądowej pozycji dotyczącej sztucznej inteligencji.

LITERATURA

[1] Bolc L. (ed.): Natural Language Communication with Computers. Lecture Notes in Computer Science, No. 63, Carl Hanser Verlag, Monachium, Wiedeń, 1980
[2] Bolc L. (ed.): Natural Language Parsing Systems. Springer-Verlag, Berlin, 1987
[3] Bolc L., Cichy M., Różańska L.: Przetwarzanie języka naturalnego. WNT, Warszawa, 1982
[4] Bolc L., Jarke M.: Cooperative Interfaces to Information Systems. Sringer-Verlag, Berlin, 1986
[5] Johnson T.: Natural language computing – the commercial applications. Ovum Ltd, London, 1985
[6] Lindsay P. H., Norman D. A.: Procesy przetwarzania informacji u człowieka. Wprowadzenie do psychologii. PWN, Warszawa 1984
[7] Mykowiecka A.: Przegląd systemów automatycznej generacji tekstów w języku naturalnym. Prace IPI PAN, Warszawa, listopad 1987
[8] Sawicka A.: Projekt i implementacja słownika języka polskiego w systemach z dostępem w języku naturalnym. Praca magisterska, Uniwersytet Warszawski, 1987
[9] Schank R. C., Childers P.: The Cognitive Computer. On Language, Learning and Artificial Intelligence. Addison Wesley Publishing Company, Reading (MA), 1985
[10] Szpakowicz S.: Formalny opis składniowy zdań polskich. Wydawnictwo UW, Warszawa, 1983
[11] Tomczak D.: Przegląd metod komputerowej analizy syntaktycznej języka naturalnego. Uniwersytet Adama Mickiewicza, Poznań, 1985
[12] Winograd T.: Language as a Cognitive Process. Addison Wesley Publishing Company, Reading (MA), 1985.

Uproszczone specyfikowanie systemów współbieżnych

dokończenie ze s. 10

Czy to wszystko wystarcza do dowodu, że realizacja z wydruku 3 jest poprawna dla specyfikacji z wydruku 4?

Tak, ponieważ obydwa systemy mają identyczne sprzężenia. Jest oczywiste, że akcje sprzężenia z wydruku 3, polegające na wywołaniu procedury i udostępnieniu wartości, poprawnie realizują akcje z wydruku 4, gdyż są to te same akcje. Dlatego wystarczy wykazać, że wewnętrzne funkcje stanu z wydruku 4 są poprawnie zrealizowane na wydruku 3, co uczyniono.

Tłumaczył i opracował
JANUSZ ZALEWSKI

LITERATURA

[1] Barringer H. et al.: A really abstract concurrent model and its temporal logic. Thirteenth Ann. ACM Symp. on Principles of Programming Languages, pp. 173–183, ACM, January 1986
[2] Bernstein R. A., Goodman N.: Concurency control in distributed database systems. ACM Computing Surveys, Vol. 13, No. 2, pp. 185–222, June 1981
[3] Guttag J. V., Horning J. J., Wing J. M.: Larch in Five Easy Pieces. Technical Report 5, Digital Equipment Corporation, Systems Research Center, Palo Alto (CA), July 1985
[4] Hoare C. A. R.: Communicating sequential processes. Comm. of the ACM, Vol. 21. No. 8, pp. 666–677, August 1978
[5] Lam S. S., Shankar A. U.: Protocol verification via projections. IEEE Trans. on Communication, Vol. 10, No. 4, pp. 325–342, July 1984.

HENRYK GIZDOŃ
ADAM PAWLAK
WŁODZIMIERZ WRONA
Instytut Elektroniki
Politechnika Śląska
Gliwice

Język opisu sprzętu VHDL – podstawowe mechanizmy (1)

W poprzednim artykule [2] przedstawiono definicję jednostki projektowej wraz z jej sprzęgiem, ciałem i deklaracją konfiguracji ciała. Omówiono zastosowanie jednostki projektowej do opisu zarówno struktury, jak i zachowania (opisu behawioralnego) wybranego układu cyfrowego. Krótko przedstawiono pojęcia typu, obiektu, podprogramu i pakietu. W tym artykule szerzej wyjaśniono te pojęcia prezentując dodatkowo szeroki wachlarz instrukcji. Opracowanie niniejsze jest oparte na wersji 7.2 języka VHDL.

TYPY JĘZYKA VHDL

Obiekty występujące w języku opisu sprzętu VHDL mają przyporządkowany określony typ. Typ stanowi zbiór wartości o wspólnych charakterystycznych cechach identyfikowany przez nazwę, przez którą obiekty dziedziczą cechy typu. Dla wybranego podzbioru wartości danego typu można zdefiniować podtyp. W języku VHDL zdefiniowano pierwotnie następujące typy i podtypy [8]:
BIT – wartości „0" i „1",
BIT_VECTOR – tablica typu BIT,
INTEGER – liczby całkowite określone przez implementację,
REAL – liczby rzeczywiste określone przez implementację,
BOOLEAN – wartości logiczne FALSE i TRUE,
CHARACTER – znaki kodu ASCII,
STRING – tablice znaków kodu ASCII,
SEVERITY_LEVEL – poziomy ważności komunikatu,
POSITIVE – liczby całkowite większe od zera,
NATURAL – liczby całkowite nieujemne.

Projektant może również definiować nowe typy i podtypy. Typy definiowane przez użytkownika można podzielić na skalarne i złożone. Typy skalarne dzielą się na wyliczeniowe, numeryczne i fizykalne. Typ wyliczeniowy jest definiowany przez wartości nienumeryczne. Korzystając z pierwotnie zdefiniowanych typów wyliczeniowych (tj. BIT, BOOLEAN i CHARACTER) projektant może łatwo zdefiniować własny typ wyliczeniowy. Przykładem może być typ definiujący zbiór elementów logiki czterowartościowej:

```
type LOG_4 is ('0', '1', 'Z', 'E');
```

Typ numeryczny jest oparty na typie rzeczywistym i całkowitym. Projektant może zdefiniować nowy typ numeryczny, specyfikując zakres wartości należący do zdefiniowanych już typów REAL i INTEGER, z określonymi na nich standardowymi operacjami. Przykłady typów numerycznych zdefiniowanych przez użytkownika:

```
type POZIOM_NAP is -5.5 to +5.5;
type INT_64K is -65536 to 65535;
```

Typ fizykalny umożliwia wyrażenie wielkości będących jednostkami miary. Deklaracja typu fizykalnego specyfikuje zbiór jednostek, opartych na jednostce podstawowej, mierzących tę samą wielkość. Przykładem typu fizykalnego jest typ VOLT:

```
type VOLT is range 0 to 1E25
  units
    nV ;                -- nanovolt
    μV = 1000 nV;       -- mikrovolt
    mV = 1000 μV;       -- milivolt
     V = 1000 mV;       -- volt
    kV = 1000 V;        -- kilovolt
end units;
```

Typy złożone składają się z elementów typów skalarnych lub innych typów złożonych. Wyróżnia się dwie klasy typów złożonych: tablice i rekordy. Typ tablicowy składa się z elementów tego samego typu. Dostęp do każdego elementu tablicy jest możliwy przez użycie jednego lub wielu indeksów. Przykłady deklaracji tablic:

```
type TABLICA_RZECZ is array (1 to 10) of REAL;
type LOG_4VECTOR is array (NATURAL range< >) of LOG_4;
```

Typ TABLICA_RZECZ deklaruje tablicę dziesięciu elementów typu REAL, a LOG_4VECTOR – tablicę elementów logiki czterowartościowej.

Typ rekordowy składa się z elementów różnych typów. Każdy element rekordu jest dostępny przez specyfikację jego nazwy wraz z kropką i nazwą elementu składowego. Przykład deklaracji rekordu:

```
type PARAMETRY_ELEKTRYCZNE is record
  NAPIĘCIE : VOLT ;
  NATĘŻENIE : AMPER ;
  REZYSTANCJA : OHM ;
end record;
```

KLASY OBIEKTÓW

W języku VHDL wyróżnia się trzy klasy obiektów: stała (ang. *constant*), zmienna (ang. *variable*) i sygnał (ang. *signal*). Dwie pierwsze są typowe dla języków programowania, trzecia wprowadza do języka obiekty o właściwościach charakterystycznych dla fizycznych sygnałów w sprzęcie. Z sygnałem jest związana przede wszystkim jego historia, pamiętana w specjalnych elementach zwanych nośnikami (ang. *drivers*). Deklarując obiekt projektant nie tylko określa jego identyfikator i typ, lecz także – w zależności od klasy obiektu – może zdefiniować jego początkową wartość, przyporządkować atrybut, wreszcie przyporządkować funkcję. Obiekt deklaruje się przez wyspecyfikowanie nazwy obiektu, a także klasy i typu danych tego obiektu. Przykładem mogą być następujące deklaracje:

```
constant STAŁA: LOG_4VECTOR: = "01";
variable ZMIENNA: LOG_4VECTOR (BUS'RANGE);
signal SYGNAŁ: BIT;
```

Sygnały są obiektami, których wartości, podobnie jak wartości zmiennych, mogą być zmieniane. Sygnały dzielą się na jednolite (ang. *atomic*) i rozdrobnione (ang. *granular*). Sygnał jednolity wyróżniony słowem zastrzeżonym *atomic* może zmieniać wartość tylko w całości, tzn. jeżeli składa się z podelementów, to nie mogą one zmieniać się niezależnie, bez oddziaływania na inne wartości elementów całego zbioru. Przykładem może być deklaracja sygnału OBWÓD_1 modelującego elektryczne zależności prawa Ohma:

```
signal OBWÓD_1: atomic PARAMETRY_ELEKTRYCZNE;
```

Sygnał OBWÓD_1, będący sygnałem jednolitym typu PARAMETRY_ELEKTRYCZNE, może zmieniać swój stan zachowując wyrażoną prawem Ohma zależność między napięciem, natężeniem i rezystancją. Każda próba zmiany jednego z elementów sygnału OBWÓD_1 jest sygnalizowana jako błąd.

Sygnał rozdrobniony składa się z elementów, których wartości zmieniają się niezależnie.

Sygnały są obiektami, które różnią się od zmiennych liczbą związanych z nimi wartości oraz sposobem zmiany tych wartości. Zmienna reprezentuje pojedyńczą wartość, sygnał natomiast – jedno- lub wieloelementowy – zbiór takich wartości. Zbiór wartości sygnału jest przechowywany w nośnikach sygnału. Wartość zmiennej i sygnału może być zmieniana instrukcją przypisania. Jeśli sygnał jest związany z większą liczbą nośników (występuje w kilku instrukcjach procesu), to wartość bieżąca sygnału jest obliczana z wartości wszystkich jego nośników za pomocą funkcji arbitra magistrali (ang. *bus resolution function*). Funkcja arbitra magistrali jest definiowana przez projektanta w deklaracji sygnału lub w deklaracji typu i podtypu sygnału (w takim wypadku każdy deklarowany sygnał tego typu dziedziczy tę funkcję). W języku VHDL połączenie wielu jednostek projektowych lub ich składników może być modelowane sygnałem z przypisaną funkcją arbitra magistrali. Przykładem może być deklaracja sygnału rozdrobnionego ZŁĄCZE typu MAGISTRALA:

```
type VECTOR_3STAN is atomic KONFLIKT_WY
LOG_4 VECTOR;
type MAGISTRALA is record
  ADRES: atomic KONFLIKT_WY LOG_4VECTOR;
  DANE: VECTOR_3STAN;
  STEROWANIE: BIT_VECTOR;
end record;
signal ZŁĄCZE: MAGISTRALA;
```

Wartości elementów sygnału rozdrobnionego ZŁĄCZE (sygnał ADRES, DANE STEROWANIE) mogą zmieniać się niezależnie. Wartości sygnałów jednolitych ADRES i DANE (elementy tablicy LOG_4VECTOR) są obliczane przy użyciu funkcji arbitra magistrali KONFLIKT_WY. Funkcja ta jest przyporządkowana sygnałowi ADRES w jego deklaracji, a sygnałowi DANE – w deklaracji typu sygnału VECTOR_3STAN.

Sygnały różnią się od zmiennych również tym, że w trakcie wystąpienia instrukcji przypisania uwzględniają uwarunkowania czasowe. Przyszłe wartości sygnałów zostają przyporządkowane nośnikom a następnie, w określonym momencie, z wartości nośników wyznacza się wartości sygnałów.

SKŁADNIKI

Opis strukturalny dowolnego układu w języku VHDL zawiera zarówno składniki biblioteczne, jak i definiowane przez użytkownika. Składniki projektowanego układu są wprowadzane do opisu projektu przez ich deklaracje [6], przy czym deklaracja składnika jednostki projektowej zawiera: nazwę składnika, listę portów i listę parametrów formalnych. Przykładem może być deklaracja bramki XOR wchodzącej w skład ciała sumatora:

```
component XOR_GATE
  port (A, B: in BIT; C: out BIT);
  generic (X_COORD: X_INTEGER; Y_COORD; Y_INTEGER);
```

Zadeklarowany składnik jednostki projektowej może być konkretyzowany w jednostce projektowej dowolnie wiele razy. W jednostce projektowej sumatora występują dwie bramki XOR X1 i X2:

```
X1: XOR_GATE port (X, Y, S1) ;   -- instrukcje konkretyzacji
                 generic (10, 20) ;
X2: XOR_GATE port (S1, CIN, SUM) ;
                 generic (20, 20) ;
```

Z pojęciem składnika wiąże się także pojęcie konfiguracji ciała. Konfiguracja ciała umożliwia przypisanie dowolnej jednostki projektowej wybranemu składnikowi ciała.

INSTRUKCJE JĘZYKA VHDL

Podobnie jak w tradycyjnych językach programowania, realizacja programu w języku VHDL polega na wykonywaniu jego instrukcji. Instrukcje te mogą:
- zmieniać wartości zmiennych i sygnałów,
- tworzyć algorytm sterowania zmianą wartości danego obiektu,
- sygnalizować błędy lub monitorować przebieg programu,
- opisywać działanie układu cyfrowego lub poszczególnych jego składników.

Instrukcje sekwencyjne

Instrukcje sekwencyjne są wykonywane zgodnie z ich porządkiem leksykograficznym w programie. Mogą występować w instrukcjach procesu lub podprogramach. Wyróżnia się następujące instrukcje sekwencyjne: przypisania, zezwolenia, zabronienia komunikatu (o niespełnieniu założenia), instrukcje sterujące oraz instrukcję wywołania procedury i instrukcję pustą (ang. *null*). Instrukcje te przypisują wartości sygnałom i zmiennym, sterują procesem, sprawdzają i sygnalizują błędy niespełnienia zadanych założeń, sterują wystąpieniem innych instrukcji oraz mogą wywoływać podprogramy. Instrukcja pusta nie wywołuje żadnych zmian w programie.

Instrukcja przypisania jest używana do przyporządkowania wartości zarówno zmiennych, jak i sygnałom. Przykładem mogą być następujące instrukcje:

- instrukcje przypisania zmiennej:

```
REZULTAT: = 'E';
PAM (WART_NAT (AD_B)) : = D_WE;
```

- instrukcje przypisania sygnału:

```
ADRES < = X"0011ABCE"; -- wartość szesnastkowa
PRZERWANIE 1 < = '1' after 2 ms, '0' after 4 ms;
```

W powyższych przykładach zilustrowano syntaktyczną różnicę między instrukcją przypisania zmiennych a instrukcją przypisania sygnałów. Po pierwsze, w instrukcji przypisania zmiennej używa się symbolu „:=", a w instrukcji przypisania sygnału – symbolu „<=". Po drugie, dla zmiennych instrukcja przypisania ma tylko jeden obiekt przypisania (przeznaczenia) po lewej stronie symbolu „:=", instrukcja przypisania sygnału natomiast może mieć wiele takich obiektów przypisania. Instrukcja przypisania zmiennej może mieć tylko jedno wyrażenie po prawej stronie symbolu instrukcji, a instrukcja przypisania sygnału może składać się z wielu wyrażeń z przyporządkowanymi im opóźnieniami. Relacje czasowe sygnałów przeznaczenia są definiowane w postaci przebiegu czasowego reprezentującego przyszłe wartości sygnałów. Słowo zastrzeżone **after** po prawej stronie instrukcji przypisania może nie występować (tak jak w wypadku sygnału ADRES). Powyższy sygnał ADRES można zapisać następująco uwzględniając opóźnienie 0 ns:

```
ADRES < = X"0011ABCE" after 0 ns;
```

Opóźnienie to w rzeczywistości jest interpretowane jako małe opóźnienie większe od zera, które dodane do bieżącego czasu symulacji nie zmienia go.

Instrukcja zezwolenia występuje w procesie, który w trakcie realizacji swoich instrukcji staje się czuły na zmianę wartości sygnałów lub zmiennych znajdujących się na liście instrukcji zezwolenia (występujących po słowie zastrzeżonym **enable**). Na początku procesu symulacji wszystkie sygnały znajdujące się na liście instrukcji **enable** są w stanie „zezwolenia". Instrukcja zezwolenia współdziała z **instrukcją zabronienia** (ang. *disable*), będącą jej uzupełnieniem. Wykonanie instrukcji zabronienia w instrukcji procesu powoduje „nieczułość procesu" na zmiany wartości sygnałów lub zmiennych znajdujących się na liście instrukcji zabronienia. Przykładem użycia instrukcji zezwolenia i zabronienia może być opis zachowania pamięci dwuportowej RAM podany dalej.

Instrukcja komunikatu służy do sygnalizowania błędów lub monitorowania przebiegu programu. Instrukcja ta składa się z wyrażenia logicznego, klauzuli raportu (ang. *report*) i klauzuli ważności (ang. *severity*), które reprezentują warunek sygnalizacji raportu, treść raportu oraz poziom ważności komunikatu (błąd, ostrzeżenie itp.). Instrukcja ta sprawdza, czy wyspecyfikowany warunek jest prawdziwy i monitoruje fakt jego niespełnienia. W instrukcji tej można zdefiniować dowolny raport określony dla danego warunku. Jeżeli brak jest raportu w instrukcji komunikatu, to w razie niespełnienia założenia pojawia się napis *„Assertion violation occurred"*. Przykładem może być instrukcja komunikatu zawierająca założenie dla warunku konfliktu na magistrali trójstanowej (Y/="EEEE") wraz z raportem o błędzie: „Konflikt na magistrali" (przy założeniu, że konflikt występuje na wszystkich bitach magistrali jednocześnie).

```
assert (Y/="EEEE")
 report
  „Konflikt na magistrali Y"
 severity ERROR;
```

Instrukcje sterujące, które występują w języku VHDL są następujące: warunkowe **if** i **case**, pętla **loop**, następnik **next**, wyjście **exit** oraz powrót **return**. Instrukcja **if** powoduje warunkowe wykonanie innych instrukcji. Instrukcja **case** powoduje wybranie jednej z grup alternatywnych sekwencji instrukcji. Instrukcja **loop** zawiera ciąg instrukcji, które mogą powtarzać się jeden lub więcej razy (zależnie od rodzaju instrukcji pętli). Jeśli w pętli wystąpiło słowo zastrzeżone **while**, to realizacja instrukcji wewnątrz pętli trwa dopóty, dopóki warunek pętli jest spełniony. Dla słowa kluczowego **for** liczba powtórzeń sekwencji instrukcji wewnątrz pętli zależy od zadeklarowanego zakresu tej instrukcji. Instrukcja **next** jest uzupełnieniem jednej z iteracji instrukcji pętli. Instrukcja **exit** powoduje opuszczenie pętli, w której się znajduje. Instrukcja **return** może występować w definicji funkcji, procedury lub procesu, realizując powrót do programu głównego.

Instrukcja wywołania procedury powoduje wywołanie przez wyspecyfikowane nazwy. Towarzyszą jej parametry aktualne zamiast zadeklarowanych parametrów formalnych. Jeżeli w instrukcji wywołania procedury brak jest jakiegoś parametru aktualnego, to zamiast niego jest przyjmowana wartość domyślna (ang. *default*).

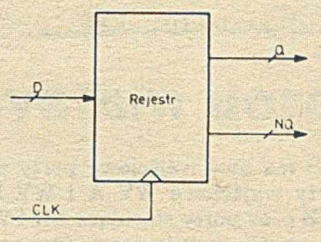
Instrukcje współbieżne

Instrukcje należące do tej klasy są (z punktu widzenia projektanta) realizowane współbieżnie, a ich porządek leksykograficzny nie ma znaczenia. Wprowadzono następujące instrukcje współbieżne: bloku, procesu, współbieżnego przypisania sygnału, współbieżnego komunikatu, konkretyzacji oraz powielania. Podstawowymi instrukcjami współbieżnymi są instrukcje bloku oraz procesu.

Instrukcja bloku służy do opisu ciała jednostek projektowych. Składa się z części deklaracyjnej i części zawierającej instrukcje współbieżne, oddzielone słowem zastrzeżonym **begin**. W części deklaracyjnej sę zawarte deklaracje stałych, zmiennych, typów, sygnałów oraz składników reprezentujących inne jednostki projektowe.

Instrukcja bloku może zawierać wyrażenie dozoru (ang. *guard*), stosowane do synchronizacji współbieżnych instrukcji przypisania znajdujących się wewnątrz bloku. Wyrażenie to jest wyrażeniem logicznym występującym w części deklaracyjnej po słowie zastrzeżonym **block**. Wprowadzając to wyrażenie do instrukcji bloku, synchronizuje się tylko instrukcje współbieżnego przypisania sygnału zawierające słowo zastrzeżone memoried, umieszczone po symbolu „<=". Instrukcje takie przypisują nową wartość sygnałowi wtedy i tylko wtedy, gdy wyrażenie dozoru jest prawdziwe. W przeciwnym razie instrukcja przypisania sygnału nie zmienia wartości danego sygnału. Jeśli brak jest wyrażenia dozoru, to przyjmuje się, że wyrażenie to jest zawsze prawdziwe.

Przykładem zastosowania instrukcji bloku może być opis rejestru z wpisem równoległym na narastającym zboczu sygnału zegarowego (rys. 1.), przedstawiony na wydruku 1. Sygnał **S** jest sygnałem wewnętrznym rejestru. Wartość tego sygnału zmienia się tylko wówczas, gdy wyrażenie dozoru jest prawdziwe. Zmiana wartości sygnałów **Q** i **NQ** nie zależy wprost od wartości wyrażenia dozoru.



Rys. 1. Rejestr z wpisem równoległym

```
-------------------------------------------------------------------
block (CLK = "1" and not CLK'STABLE) -- wyrazenie dozoru
     signal S : BIT_VECTOR;                -- sygnal wewnetrzny
begin
     S <= memoried D;                      -- synchronizowany
     Q <= S after 10 ns;                   -- nie synchronizowany
     NQ <= not S after 10 ns;              -- nie synchronizowany
end block;
-------------------------------------------------------------------
```

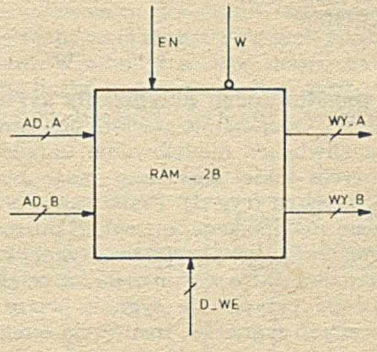
Wydruk 1

Instrukcja bloku może być synchronizowana poziomem lub zmianą sygnału występującego w wyrażeniu dozoru (np. zboczem narastającym sygnału zegarowego CLK w powyższym przykładzie). Do tego celu służy atrybut STABLE, który umożliwia określenie, czy dany sygnał zmienił swoją wartość.

Współbieżną instrukcją bloku może być w szczególności inna instrukcja bloku. Pozwala to na hierarchiczną budowę jednostek projektowych z możliwością zagnieżdżania bloków, co z kolei umożliwia dekompozycję funkcji jednostki projektowej na podfunkcje, a także struktury na podstruktury.

Instrukcja procesu definiuje niezależny proces, który zainicjowany wystąpieniem pewnych zdarzeń wykonuje sekwencję zapisanych w nim instrukcji. Instrukcja procesu może generować nowe zdarzenia zmieniające wartości sygnałów innych instrukcji procesu. Instrukcja ta składa się z trzech podstawowych części: listy czułości (ang. *sensitivity list*), części deklaracyjnej i ze zbioru instrukcji. Lista czułości definiuje zbiór sygnałów, których zmiana inicjuje działanie instrukcji procesu. Lista ta występuje po słowie zastrzeżonym **process**. Sygnały z tej listy muszą być zadeklarowane przed ich użyciem w danym procesie. Część deklaracyjna zawiera deklaracje lokalnych danych używanych do obliczeń w tym procesie, takich jak typy, stałe, zmienne i sygnały.

Sekwencja instrukcji definiuje algorytm zmian wartości obiektów wewnętrznych i wyjściowych w zależności od obiektów wejściowych. W zbiorze instrukcji procesu mogą występować instrukcje przypisania sygnału i zmiennej oraz inne instrukcje sekwencyjne. Semantyka instrukcji procesu umożliwia zapis kolejnych zdarzeń reprezentujących działanie projektowanej jednostki. Przykładem instrukcji procesu może być opis zachowania pamięci dwuportowej RAM (rys. 2) przedstawiony na wydruku 2.



Rys. 2. Pamięć dwuportowa

```
process (AD_A,AD_B,D_WE,EN,W)
      -- deklaracja zmiennej lokalnej PAM typu MATRYCA
      -- 16 slow danych typu BIT_VECTOR
      subtype DANE is BIT_VECTOR (3 downto 0);
      type MATRYCA is array (0 to 15) of DANE;
      variable PAM: static MATRYCA;
      begin
      WY_A <= PAM (WART_NAT (AD_A));
      WY_B <= PAM (WART_NAT (AD_B));
      if (EN='1') and (W='0') then
            enable D_WE;
            PAM (WART_NAT (AD_B)) := D_WE;
      else
            disable D_WE;
      end if;
end process;
```

Wydruk 2

Pamięć RAM_2B ma dwa wejściowe porty adresowe AD_A i AD_B, dwa porty wyjściowe WY_A i WY_B, port danych wejściowych D_WE oraz porty sterujące EN i W. Zapis danych pod adres **B** jest możliwy wówczas, gdy sygnały sterujące przyjmują wartość: **W = ‚0' i EN = ‚1'.** W pozostałych wypadkach wyjściom WY_A i WY_B jest przyporządkowana zawartość komórek pamięci wybranych przez adresy **A** i **B**. Instrukcja procesu zawiera instrukcje przypisania sygnałów wyznaczające wartości wyjść WY_A i WY_B. Sygnałami wejściowymi instrukcji procesu (umieszczonymi na liście czułości) są AD_A, AD_B, D_WE, EN i W. Instrukcje zezwolenia i zabronienia sterują „czułością" procesu na zmianę sygnału danych wejściowych D_WE. W powyższym procesie korzysta się z funkcji WART_NAT, zmieniającej dane BIT_VECTOR na liczby naturalne.

Dla sygnałów magistrali istnieje w programie wiele instrukcji procesu wyznaczających wartości sygnałów. Rozważmy następujące instrukcje procesu:

```
process (WEJ_1)
begin
  DANE <= WEJ_1;
end process;
process (WEJ_2)
begin
  DANE <= WEJ_2;
end process;
```

Każda z dwóch instrukcji procesu wyznacza oddzielną, różną wartość nośnika sygnału DANE. Ponieważ istnieją dwa nośniki sygnału DANE, jego wartość musi być obliczana za pomocą funkcji arbitra magistrali (np. funkcji KONFLIKT_WY).

Współbieżna instrukcja komunikatu jest specyficzną instrukcją procesu, której lista czułości zawiera nazwy obiektów (zmiennych, sygnałów) występujących w wyrażeniu instrukcji komunikatu. Wystąpienie współbieżnej instrukcji komunikatu jest równoważne z wystąpieniem instrukcji procesu, który ją reprezentuje. Taka instrukcja procesu nie ma sygnałów wyjściowych. Wystąpienie tej instrukcji nie powoduje pojawienia się nowego zdarzenia w procesie symulacji. Jeżeli wyrażenie logiczne w instrukcji komunikatu jest fałszywe, to zadeklarowany raport jest rejestrowany. Przykładem instrukcji komunikatu może być następująca instrukcja procesu:

```
process (Y)
 assert Y/="EEEE"
  report
   „Konflikt na magistrali"
  severity ERROR,
end process;
```

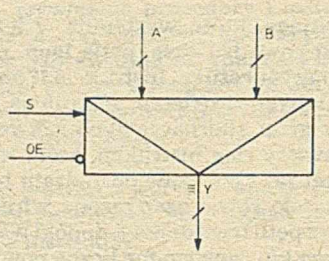
Instrukcja współbieżnego przypisania sygnału występuje w instrukcji bloku. Opcjonalnie zawiera ona słowo zastrzeżone **memoried** umożliwiające modelowanie układów z pamięcią. Przykładem użycia takiej instrukcji może być przytoczony wcześniej opis rejestru (rys. 1). Przedstawiony rejestr można opisać za pomocą równoważnej instrukcji procesu (wydruk 3).

Instrukcja współbieżnego przypisania sygnału może występować również jako instrukcja warunkowa lub selektywna. Instrukcja warunkowego przypisania sygnału wiąże przypisanie z wartością wybranych sygnałów. Instrukcja selektywnego przypisania sygnału jest stosowana wtedy, kiedy przypisanie jest związane z pojedyńczym wyrażeniem, identycznym jak dla instrukcji **case**. Przykładem zastosowania warunkowej współbieżnej instrukcji przypisania sygnału może być opis zachowania multipleksera (rys. 3), przedstawiony na wydruku 4 za pomocą bloku i równoważnej instrukcji procesu.

```
process (CLK)
      signal S : BIT_VECTOR;
begin
      if (CLK and not CLK'STABLE) then
            S <= D;
            Q <= S after 10 ns;
            NQ <= not S after 10 ns;
      end if;
end process;
```

Wydruk 3

W opisie układów z pamięcią, charakteryzującym się występowaniem słowa zastrzeżonego **memoried**, można również korzystać z instrukcji warunkowego i selektywnego przypisania sygnału.



Rys. 3. Multiplekser z wyjściem trójstanowym

```
block
      begin
      Y <= ZAPISZ_WY(A) after 10 ns when (S='0') and (OE='0')
            else
            ZAPISZ_WY(B) after 10 ns when (S='1') and (OE='0')
            else
            ZAPISZ_Z(Y'RANGE) after 12 ns;
end block;
```

```
process (A,B,S,OE)
      begin
            if S = '0' and OE = '0' then
                  Y <= ZAPISZ_WY(A) after 10 ns;
            elsif S = '1' and OE = '0' then
                  Y <= ZAPISZ_WY(B) after 10 ns;
            else
                  Y <= ZAPISZ_Z(Y'RANGE) after 12 ns;
            end if;
end process;
```

Wydruk 4

Instrukcja konkretyzacji umożliwia konkretyzację składników z zadeklarowanych wcześniej jednostek projektowych. Jednostka projektowa jest formą, wzorem dla tworzenia składników. Może ona być wywoływana dowolnie wiele razy. Każda instrukcja konkretyzacji składnika tworzy jeden składnik o określonych parametrach formalnych (rodzajowych). Przykładem może być utworzenie składnika OR1 z jednostki projektowej bramki OR:

```
OR1: OR_GATE port (S2, S3, COUT);
                  generic (20, 35);
```

Instrukcja powielania umożliwia tworzenie regularnych struktur układów wraz z połączeniem ich portów. Pozwala na konkretyzację większej liczby identycznych składników, co upraszcza zapis wielu instrukcji konkretyzacji. Można przedstawić konkretyzację bramek OR korzystając z instrukcji powielania:

```
for i=0 to 3 generate
  OR_Y:OR_GATE port (X (3*i), X (3*i + 1), X (3*i + 2), Y (i) );
end generate;
```

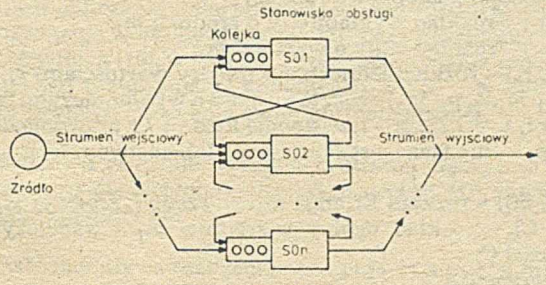
Instrukcja powielania może również zawierać mechanizm warunkowej konkretyzacji składnika. Instrukcje zawarte w instrukcji powielania nie są ograniczone liczbowo, mogą również zawierać inne instrukcje zagnieżdżone.

dokończenie na s. 22

TADEUSZ CZACHORSKI,
MAREK KOWALÓWKA
ZDZISŁAW SZCZERBIŃSKI, ANDRZEJ WILK
Zakład Systemów Automatyki Kompleksowej
Polskiej Akademii Nauk
Gliwice

AMOK – mikrokomputerowy system doradczy projektanta i analityka systemów komputerowych

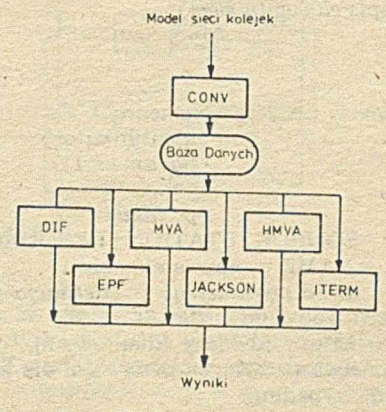
Probabilistyczne modele systemów komputerowych, przedstawiające badany system w postaci sieci stanowisk obsługi (przy czym stanowiska reprezentują elementy systemu: jednostkę centralną, pamięci zewnętrzne, terminale, a klienci – wykonywane zadania), są najczęściej stosowaną analityczną metodą oceny działania systemów obliczeniowych. Wyniki tych modeli różnią się od wartości rzeczywistych zazwyczaj nie więcej niż o 5% w wypadku oceny przepustowości systemu i nie więcej niż o 30%, gdy chodzi o predykcję czasów reakcji systemu [6]. Zastosowanie modeli, mimo ich prostoty pojęciowej, wymaga często dość złożonych obliczeń, a więc i odpowiedniego oprogramowania. W ciągu ostatnich dwudziestu lat powstało wiele pakietów obliczeniowych poświęconych modelowaniu, z których najbardziej znane to francuski QNAP [9] oraz amerykańskie RESQ [11] i BEST [2]. Analizator Modeli Kolejkowych AMOK jest opracowanym w ZSAK PAN w ramach CPBR 8.6 pakietem programowym wykonującym obliczenia dla modeli mających postać sieci stanowisk obsługi. Pakiet jest napisany w Fortranie (zawiera ponad 15 000 wierszy) i jest przeznaczony dla mikrokomputerów zgodnych z IBM PC/XT i AT. Jego głównym zastosowaniem jest modelowanie i ocena działania systemów komputerowych. Może być również wykorzystany w innych, tradycyjnych zastosowaniach teorii masowej obsługi, gdzie modelowane obiekty – np. linie produkcyjne, magazyny, stanowiska przeładunkowe – są przedstawiane jako sieć kolejek i stanowisk obsługi.



Rys. 1. Ogólna postać sieci stanowisk obsługi

Na rysunku 1 przedstawiono ogólną postać modelu analizowanego przez AMOK. Jest to sieć stanowisk obsługi o dowolnej topologii. Każde stanowisko jest opisane rozkładem czasu obsługi i regulaminem kolejki. Drogę klientów między stanowiskami w sieci określają prawdopodobieństwa przejść. Sieć może być otwarta lub zamknięta. Do sieci otwartej klienci nadchodzą strumieniami (zdefiniowanymi przez rozkłady prawdopodobieństwa czasów między pojawieniem się kolejnych klientów) i odchodzą z niej z określonym prawdopodobieństwem. W sieci zamkniętej krążą wciąż ci sami klienci. Klienci są zgrupowani w klasy, z których każda ma odmienne rozkłady czasu obsługi i charakterystyczne dla niej prawdopodobieństwa przejść między stanowiskami. Należy obliczyć mierniki efektywności pracy systemu: *średnią długość kolejki* przed każdym stanowiskiem, *średni czas czekania w kolejce*, *przepustowość stanowiska*, *stopień wykorzystania stanowiska*, a także *średni czas reakcji* całego systemu.

Nie ma uniwersalnej metody rozwiązania postawionego wyżej problemu. W zależności od przyjętych założeń co do rozkładów czasu obsługi i regulaminów szeregowania klientów w kolejkach, zadanie może być rozwiązane w sposób analityczny (dokładny lub przybliżony) – albo też jego analityczne rozwiązanie nie jest możliwe i trzeba stosować symulację cyfrową.



Rys. 2. Struktura systemu AMOK

AMOK składa się z oprogramowania, umożliwiającego użytkownikowi łatwy opis modelu w specjalnie przystosowanym do tego celu języku, oraz z modułów obliczeniowych odpowiadających najczęściej stosowanym metodom analizy sieci kolejek. Na rysunku 2 przedstawiono obecną strukturę pakietu. Moduł CONV (Konwertor) analizuje model wejściowy zapisany w języku opisu sieci kolejek [5] i w wypadku bezbłędnej analizy syntaktycznej i semantycznej, tworzy bazę danych – plik zawierający parametry modelu, z którego następnie czerpie dane wybrany moduł obliczeniowy (JACKSON, MVA itd.). Są to moduły odpowiadające metodom analitycznym; modułu symulacyjnego AMOK nie ma, choć język i baza danych pakietu są przygotowane na ewentualne dołączenie tego modułu. W tabeli 1 zawarto główne cechy sieci rozpatrywanych przez poszczególne moduły pakietu. W tabeli 2 podano maksymalne rozmiary sieci (maksymalną liczbę stanowisk obsługi, liczbę klas, maksymalną liczbę klientów dopuszczalną w sieci zamkniętej) analizo-

Tabela 1. Typy sieci analizowane przez AMOK

Moduł	Typ sieci		Liczba klas		Typ stanowisk	
	Otwarta	Zamknięta	Jedna	Wiele	BCMP	G/G/1
JACKSON	+			+	+	
MVA, HMVA		+		+	+	
EPF		+	+		+	+
ITERM		+	+		+	+
DIF	+			+		+

wanych przez wersję systemu AMOK, wygenerowaną przez autorów na mikrokomputerze IBM PC/XT z 640 KB pamięci operacyjnej.

Tabela 2. Maksymalne rozmiary sieci analizowanych przez wersję systemu AMOK wygenerowaną na IBM/PC XT (ograniczeniem w wypadku nieskończonej liczby klientów jest tylko czas obliczeń)

Moduł	Liczba źródeł	Liczba stacji	Liczba klas	Liczba klientów
JACKSON	12	12	8	–
DIF	12	12	4	–
MVA	–	30	10	
HMVA	–	30	10	
EPF	–	5	1	10
ITERM	–	10	1	50

JĘZYK OPISU MODELU

Opis sieci stanowisk obsługi w języku systemu AMOK [13] składa się z opisu poszczególnych elementów sieci: *źródeł* (w wypadku sieci otwartej), *stanowisk obsługi* i ich *kolejek* oraz *elementów biernych*, przyznawanych klientom, a później przez nich zwalnianych. Prawdopodobieństwa przejść między stanowiskami zalicza się do parametrów elementów sieci.

⟨*opis sieci*⟩ ::= ⟨*opis elementu sieci*⟩
...
⟨*opis elementu sieci*⟩

gdzie:

⟨*opis elementu sieci*⟩ ::= /⟨*typ elementu*⟩/
⟨*identyfikator*⟩
⟨*parametr 1*⟩
...
⟨*parametr n*⟩
⟨*typ elementu*⟩::=SOURCE | STATION | RESOURCE
⟨*identyfikator*⟩::=NAME = nazwa elementu
⟨*parametr*⟩::= ⟨*nazwa parametru*⟩ = ⟨*definicja parametru*⟩

Parametry, które zależą od klasy klientów, np. czas obsługi i prawdopodobieństwa przejść, są podawane dla każdej klasy oddzielnie. W tym wypadku:

⟨*parametr*⟩ ::=⟨*nazwa parametru*⟩ (⟨*pierwsza lista klas*⟩)
... =⟨*definicja parametru*⟩
⟨*nazwa parametru*⟩(⟨*k-ta lista klas*⟩) =⟨*definicja parametru*⟩

gdzie:

⟨*lista klas*⟩::= *nazwa klasy,..., nazwa klasy.*

Przyjęto następujące nazwy parametrów:

⟨*nazwa parametru*⟩::=MAX | SIZE | SCHEDULE | SERVICE | TRANSITION

określające:
MAX – maksymalną długość kolejki,
SIZE: dla stanowiska – liczbę kanałów obsługi; dla zasobu – wymiar zasobu;
SCHEDULE – regulamin szeregowania klientów;
SERVICE: dla źródła – rozkład odstępów czasu między chwilami generowania kolejnych klientów; dla stanowiska obsługi – rozkład czasu obsługi; dla zasobu – ilość pobieranego zasobu,
TRANSITION – prawdopodobieństwa przejść między stanowiskami.
Parametry MAX i SIZE przyjmują wartości całkowite i są opcjonalne. Parametr SCHEDULE podaje nazwę regulaminu szeregowania z występującą po niej, ujętą w nawiasy okrągłe, listą szeregowanych w tym elemencie sieci klas klientów:

⟨*definicja* SCHEDULE⟩::=⟨*nazwa regulaminu szeregowania*⟩ (⟨lista klas⟩

gdzie:

⟨*nazwa regulaminu szeregowania*⟩ ::=FIFO | LIFO | LIFO-PR | IS |PS | PRIOR | PRIOR-PR.

Poszczególne nazwy oznaczają:
FIFO (ang. *first-come-first-served*) naturalny regulamin kolejki, tj. kolejność obsługi zgodną z kolejnością nadejścia klientów;
LIFO (ang. *last-come-first-served*) – regulamin obsługi w kolejności odwrotnej do kolejności nadejścia klientów;
LIFO-PR (ang. *last-come-first-served-preempted-resume*) – regulamin LCFS z przerwaniem bieżącej obsługi w chwili nadejścia nowego klienta;
IS (ang. *infinite server*) – stanowisko z nieskończoną liczbą równoległych kanałów obsługi;
PS (ang. *processor sharing*) – obsługa z podziałem czasu procesora – wszyscy obecni w stanowisku klienci są obsługiwani jednocześnie, czas ich obsługi rośnie proporcjonalnie do liczby obsługiwanych klientów;
PRIOR – kolejka priorytetowa bez przerwań, wybór klienta o najwyższym priorytecie i jego obsługa następują każdorazowo po zakończeniu obsługi poprzedniego klienta;
PRIOR-PR – regulamin priorytetowy z przerwaniami; obsługa klienta jest zawieszona w chwili nadejścia klienta o wyższym priorytecie. Jeżeli parametr SCHEDULE nie występuje, to przyjmuje się, że regulamin jest typu FIFO.

Ogólną definicję parametru SERVICE stanowi zestaw oddzielonych od siebie znakiem równości definicji prostych, którymi mogą być definicje rozkładu losowego odcinków czasowych lub definicje pobrania (zwrotu) zasobu przez klienta.
⟨*definicja* SERVICE⟩ :: = ⟨*prosta definicja* SERVICE⟩
...
= ⟨*prosta definicja* SERVICE⟩

gdzie:

⟨prosta definicja SERVICE⟩ :: = ⟨*definicja rozkładu losowego*⟩|
⟨*definicja pobrania zasobu*⟩|
⟨*definicja zwrotu zasobu*⟩
⟨*definicja rozkładu losowego*⟩ :: =
⟨*nazwa rozkładu*⟩ [*parametr rozkładu,..., parametr rozkładu*]
⟨*nazwa rozkładu*⟩ :: = EXP | DET | UNI | COX | ERL | G | HIP
⟨*definicja pobrania zasobu*⟩ :: = RESERVE [*liczba porcji*]
⟨*definicja zwrotu zasobu*⟩ :: = RELEASE (*nazwa zasobu* [*liczba porcji*], (..., *nazwa zasobu* [*liczba porcji*])

Uwzględniono następujące rozkazy:
EXP $[m_1]$ – rozkład wykładniczy o średniej m_1;
DET $[m_1]$ – rozkład stały o wartości m_1;
UNI $[r_1, r_2]$ – rozkład jednostajny między wartościami r_1 i r_2;
COX $[p_1, m_1, ..., p_n, m_n]$ – rozkład Coxa z n wykładniczymi fazami; po fazie i, $i = 1,2,...,n-1$, następuje *faza* $i+1$ z prawdopodobieństwem p_{i+1} lub faza i jest z prawdopodobieństwem $1-p_{i+1}$ fazą ostatnią;
ERL $[n, m_1]$ – rozkład Erlanga n-tego rzędu o średniej m_1,
G $[m_1, C]$ – rozkład ogólny o średniej m_1 i współczynniku zmienności C;
$C = \sqrt{var}/m^2_1$;
HIP $[p_1, m_1, ..., p_n, m_n]$ – rozkład hiperwykładniczy o n równoległych wykładniczych fazach; wybierana jest tylko jedna z nich: faza i, $i = 1,..., n$, z prawdopodobieństwem p_i.
Prawdopodobieństwa przejść określa się następująco:

⟨*definicja* TRANSITION ⟩ :: = ⟨*przesłanie proste*⟩|
[*prawdopodobieństwo przesłania*] ⟨*przesłanie proste*⟩,
...
[*prawdopodobieństwo przesłania*] ⟨*przesłanie proste*⟩

gdzie:

⟨przesłanie proste⟩ :: = nazwa elementu sieci : nazwa klasy klientów |
nazwa elementu sieci |
OUT.

Występująca powyżej *nazwa elementu sieci* jest nazwą elementu STATION lub RESOURCE nadaną mu przez użytkownika.

Jeżeli podczas przesłania klient zmienia klasę, do której należy, to fakt ten zaznacza się w przesłaniu prostym podając za znakiem dwukropka nazwę nadawanej klasy. Symbol OUT pozwala w sposób jawny zadeklarować, że klienci opuszczają sieć.

Klasa modeli opisywanych przez język systemu AMOK jest szersza od możliwości rozwiązywania problemów przez zawarte obecnie w tym pakiecie moduły analityczne: moduły te nie analizują synchronizacji klientów w sieci ani nie uwzględniają obecności elementów biernych.

MODUŁY OBLICZENIOWE

Modelowanie w systemie AMOK polega na obliczeniu wskaźników efektywności pracy charakteryzujących stan ustalony (stan równowagi stochastycznej) sieci stanowisk obsługi. Metody analityczne dostępne obecnie w tym systemie są metodami dokładnymi lub przybliżonymi. Po ustaleniu założeń dotyczących postaci strumieni wejściowych, rozkładów czasów obsługi i regulaminów kolejek metody dokładne dają ścisłe matematycznie rozwiązania; w metodach przybliżonych upraszcza się założenia lub też dokonuje przybliżeń w trakcie obliczeń.

Metody dokładne są zawarte w modułach JACKSON i MVA; są one związane z najogólniejszym modelem analitycznym, zwanym od nazwisk autorów modelem BCMP [1]. Model ten zakłada wiele klas klientów, sieci mieszane (otwarte dla pewnych klas klientów, zamkniętych dla pozostałych) oraz stanowiska obsługi z regulaminem FIFO, PS, IS lub LIFO-PR.

W stanowiskach z regulaminem FIFO czas obsługi ma rozkład wykładniczy, jednakowy dla wszystkich klas klientów; w pozostałych stanowiskach jest dopuszczalny rozkład typu Coxa, różny dla każdej klasy klientów.

Zaletą rozkładów Coxa jest to, że przy ich użyciu można przybliżyć z dowolną dokładnością każdy występujący w praktyce rozkład prawdopodobieństwa. Największym problemem numerycznym modelu BCMP jest obliczanie stałej normalizacyjnej, służącej do skalowania prawdopodobieństw stanów: liczba stanów szybko wzrasta wraz ze wzrostem liczby stanowisk i liczby klientów w systemie i na przykład liczba stanów zamkniętej sieci dziesięciu stanowisk, w której krąży dziesięciu klientów przekracza 92 tysiące. Dlatego AMOK analizuje dwa modele o zawężonych nieco założeniach. Moduł JACKSON rozwiązuje otwartą sieć BCMP z czasami obsługi niezależnymi od liczby klientów w kolejce – w tym wypadku nie zachodzi konieczność obliczania stałej normalizacyjnej; moduł MVA natomiast rozwiązuje zamkniętą sieć BCMP przy użyciu algorytmu wartości średnich [10], w którym, przez zastosowanie procedury rekurencyjnej rozwiązującej kolejno sieć z 1,2,..., N klientami unika się jawnego obliczania stałej normalizacyjnej.

Moduł HMVA realizuje wersję przybliżoną algorytmu MVA i jest stosowany dla dużych zamkniętych sieci BCMP, zawierających wielu klientów [4].

Pozostałe moduły są realizacją następujących metod przybliżonych, umożliwiających analizę dużych i bardziej ogólnych sieci, w szczególności sieci zawierających stanowiska z regulaminem obsługi FIFO i dowolnym rozkładem czasu obsługi, odmiennym dla każdej klasy klientów:

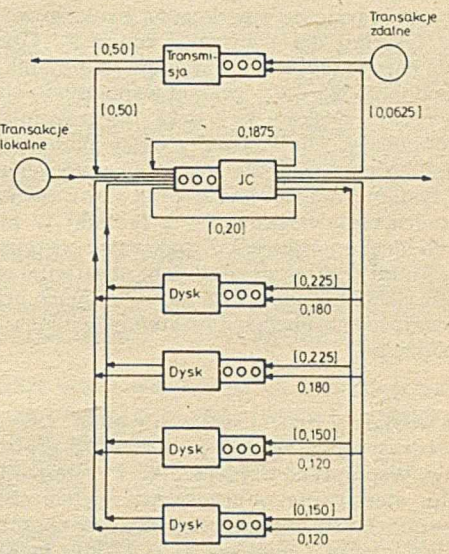
EPF – rozszerzona forma iloczynowa, metoda aproksymująca rozkład $p(n)$ liczby klientów w każdym stanowisku sieci przez $p(n)$ dla pojedynczego stanowiska $M|G|1|N$ [12];

DIF – aproksymacja dyfuzyjna, gdzie proces $N(t)$ – liczba klientów obecnych w stanowisku w chwili t jest zastępowany procesem dyfuzji; $p(n)$ jest aproksymowane funkcją gęstości prawdopodobieństwa procesu dyfuzji otrzymaną jako rozwiązanie równania dyfuzji z odpowiednio dobranymi współczynnikami i warunkami brzegowymi, patrz np. [7];

ITERM – metoda iteracyjna z zastosowaniem twierdzenia Nortona do sieci stanowisk obsługi, każde stanowisko „widzi" resztę sieci jako źródło klientów, generując strumień Poissona z parametrem zależnym od liczby klientów obecnych w rozpatrywanym stanowisku [8].

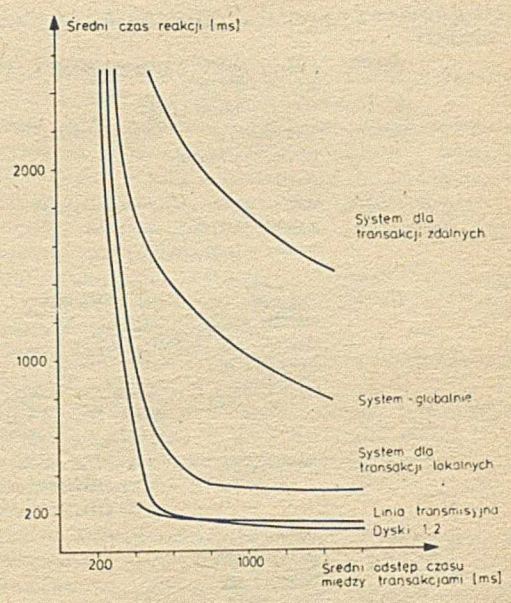
MODEL SYSTEMU TRANSAKCYJNEGO

Na rysunku 3 przedstawiono model sieci otwartej sporządzony dla transakcyjnego systemu komputerowego. Jest to uproszczony ze względów dydaktycznych model rzeczywistego systemu prowadzącego ewidencję gospodarki materiałowej, energetycznej, zamówień i ich realizacji oraz sprawy osobowe załogi w dużym zakładzie przemysłowym. System wykonuje zadania zdalne, napływające linią transmisyjną (pierwsza klasa klientów) i zadania lokalne, generowane z końcówek interakcyjnych przy systemie (druga klasa klientów). Transakcje obu typów różnią się częstością korzystania z dysków, co ma swój wyraz w zaznaczonych na rysunku wartościach prawdopodobieństw przejść między stanowiskami (wartości prawdopodobieństw dla transakcji zdalnych są ujęte w nawiasy).



Rys. 3. Model systemu

Zakłada się, że czasy transmisji obu typów transakcji i czasy ich obsługi w dyskach można przybliżyć rozkładami wykładniczymi, których wartość średnia wynosi dla linii transmisyjnej $S_6 = 100\ ms$, a dla dysków $S_1 = S_2 = S_3 = S_4 = 70\ ms$.



Rys. 4. Średnie czasy systemu i poszczególnych stanowisk dla transakcji zdalnych, w funkcji intensywności nadejść tych transakcji

Czasy obsługi transakcji w jednostce centralnej są opisane rozkładami Coxa, których wartość średnia wynosi $S_5^{(1)} = 9\ ms$

dla pierwszego typu transakcji i $S_5^{(2)} = 5\ ms$ dla drugiego typu transakcji. Parametry tych rozkładów są podane w opisie modelu na rys. 4. Dostęp do jednostki centralnej odbywa się zgodnie z regulaminem podziału procesora. Średnio czas między nadejściem transakcji lokalnych $1/\lambda_2 = 500\ ms$.

Należy zbadać pracę systemu dla powyższych danych przy zmieniającym się strumieniu λ_1 transakcji zdalnych.

Zapis modelu w języku pakietu przedstawiono na wydruku. Wyniki obliczeń modułu JACKSON przeprowadzonych dla $1/\lambda_1 = 500\ ms$ pominięto (ze względu na brak miejsca – przyp. red.) przedstawiając (rys. 4) średnie czasy reakcji systemu i poszczególnych stanowisk dla transakcji zdalnych, w funkcji intensywności przychodzenia tych transakcji.

Najdłuższe czasy reakcji występują dla dysków 1 i 2 – to one są „wąskim gardłem" systemu i przyczyną stosunkowo długiego czasu reakcji systemu dla transakcji zdalnych. Czas reakcji systemu rośnie bardzo szybko ze wzrostem strumienia zdalnych transakcji – podobnie jak czasy reakcji wspomnianych dysków. Wzrost czasu reakcji dysków 3 i 4 oraz linii transmisyjnej jest znacznie łagodniejszy. Dla czasów między nadejściami transakcji zdalnych krótszych niż 300 ms (intensywności większych od 3,33 [1/s]), system traci stabilność (brak stanu stacjonarnego) z powodu przeciążenia dysków 1 i 2 (kolejki w nich rosną teoretycznie do nieskończoności). Zwiększając odstępy czasowe między nadejściami transakcji zdalnych, obserwuje się istotne skrócenie czasu reakcji dysków 1 i 2; przy odstępach przekraczających 700 ms najbardziej obciążonym stanowiskiem w sieci staje się linia transmisyjna.

* * *

Przewiduje się dalszy rozwój systemu AMOK. Obecnie przygotowuje się wersję wykorzystującą możliwości graficzne mikrokomputerów. Współpraca użytkownika z systemem (definiowanie modelu, sterowanie obliczeniami, wybór obliczanych wielkości, sposób ich prezentacji) odbywa się w sposób interakcyjny przy użyciu myszki i widocznego na ekranie menu. Wyniki mogą być wyświetlane w postaci graficznej.

Planuje się dołączenie dalszych modułów obliczeniowych (implementacja algorytmu MVA w wypadku regulaminów priorytetowych, modele z synchronizacją klientów w sieci, uwzględnienie zasobów pasywnych). AMOK powinien stać się systemem ekspertowym wspomagającym pracę analityka systemów komputerowych lub innych systemów opisywalnych w języku modeli masowej obsługi. Jest to również narzędzie dydaktyczne, pomocne przy zaznajamianiu się studentów z kolejkowymi modelami systemów komputerowych i problemami numerycznymi, pojawiającymi się podczas rozwiązywania takich modeli.

```
SOUR przyklad sieci otwartej
•DESCRIPTION•
          /SOURCE/   NAME = DISTTERM  ! terminale zdalne !
                     SERVICE = EXP[500]
                     TRANSIT = TRANSM:DISTANT
          /SOURCE/   NAME = LOCTERM   ! terminale lokalne !
                     SERVICE = EXP[500]
                     TRANSIT = CPU:LOCAL
          /STATION/  NAME = TRANSM    ! linia transmisyjna !
                     SERVICE = EXP[100]
                     TRANSIT = [0.5] CPU,
                               [0.5] OUT
          /STATION/  NAME = CPU    ! jednostka centralna !
                     SERVICE(:DISTANT) = COX[1.6,0.5,6]
                     SERVICE(:LOCAL) = COX[1.4,0.2,5]
                     SCHEDULE = PS
                     TRANSIT(:DISTANT) = [0.1875] CPU,[0.0625] TRANSM,
                                         [0.225] DISK1,[0.225] DISK2,
                                         [0.15] DISK3,[0.15] DISK4
                     TRANSIT(:LOCAL) =   [0.2] CPU,[0.18] DISK1,
                                         [0.18] DISK2,[0.12] DISK3,
                                         [0.12] DISK4,[0.2] OUT
          /STATION/  NAME = DISK1
                     SERVICE = EXP[70]
                     TRANSIT = CPU
          /STATION/  NAME = DISK2
                     COPY = DISK1
          /STATION/  NAME = DISK3
                     COPY = DISK1
          /STATION/  NAME = DISK4
                     COPY = DISK1
•END•
```

Zapis modelu w języku systemu AMOK

Pierwsza wersja systemu AMOK, wyposażona dodatkowo w moduł optymalizacyjny i przeznaczona dla systemu ODRA 1305, jest rozpowszechniana przez Zakład Systemów Automatyki Kompleksowej PAN w Gliwicach; opisana tu wersja pakietu, przeznaczona dla mikrokomputerów zgodnych z IBM PC, a w szczególności dla mikrokomputera ComPAN-16, jest rozpowszechniana przez ZSAK PAN oraz Zakłady Urządzeń Komputerowych MERA-ELZAB w Zabrzu.

LITERATURA

[1] Baskett F., Chandy M., Muntz R., Palacios J.: Open, Closed and Mixed Networks of Queues with Different Classes of Customers. J. ACM, Vol. 22, No. 2,1975

[2] Best/1 – Product Description, BGS Systems, BE 77-010-2 Lincoln, Mass. 1977

[3] Buzen J. P.: Computational Algorithms for Closed Queueing Networks with Exponential Servers. Comm. ACM, Vol. 16, No. 9, 1973

[4] Chandy K. M., Neuse D.: Linearizer – A Heuristic Algorithm for Queueing Network Models of Computing Systems. Comm. ACM, Vol. 25, No. 2, 1982

[5] Czachórski T., Kowalówka M., Szczerbiński Z., Wilk A.: Analiza pracy systemów komputerowych przy wykorzystaniu modeli probabilistycznych. Pakiet programowy AMOK. Politechnika Śląska, Gliwice 1987

[6] Denning P. J., Buzen J. P.: The Operational Analysis of Queueing Network Models. Computing Surveys, Vol. 10, No. 4, pp. 225-261

[7] Gelenbe E., Pujolle G.: The Behaviour of a Single Queue in a General Queueing Network. Acta Informatica, Fasc. 7, 1976

[8] Marie R.: An Approximate Analytical Method for General Queueing Networks. IEEE Trans. on Software Eng., Vol. SE-5, No. 5, 1979

[9] Potier D.: New User's Introduction to QNAP2. Rapport Technique, No. 40, INRIA, Roquencourt 1984

[10] Reiser M., Lavenberg S. S.: Mean Value Analysis of Closed Multichain Queueing Networks. J. ACM, Vol. 27, No.2, 1980

[11] Sauer C. H., Mac Nair E. A., Kurose J. F.: The Research Queueing Package Version 2 – Introduction and Examples. IBM Research Report RA-138, Yorktown Heights, New York 1982

[12] Shum A. W.: Queueing Models for Computer Systems with General Service Time Distribution. Ph. D. Thesis Harvard University, Cambridge, Mass. 1976

[13] Wilk A.: QNDEL – Język opisu sieci kolejek. Podstawy Sterowania, Vol. 15, No. 4, pp. 283-296, 1985.

Język opisu sprzętu VHDL

dokończenie ze s. 18

Język VHDL zawiera bogaty zbiór typów pierwotnych oraz umożliwia użytkownikowi deklarowanie własnych typów. Klasa obiektów języka jest rozszerzona o nowe pojęcie – sygnał, który umożliwia dogodną notację przepływu danych między składnikami układu. Dodatkową jednostką języka jest nośnik, umożliwiający zapamiętywanie kolejnych wartości sygnałów. Zbiór instrukcji języka umożliwia wyrażenie pełnego opisu funkcjonalnego i strukturalnego projektu układu. Na uwagę zasługuje nowa interesująca koncepcja instrukcji procesu, umożliwiająca symulację procesową układu cyfrowego [5].

W następnej części artykułu zostaną przedstawione szerzej pojęcia parametru formalnego, podprogramu, pakietu oraz dyrektywy. Dodatkowo zostanie przedstawiona deklaracja czasu, związana z procesem symulacji projektowanych układów.

LITERATURA

[1] Aylor J. H. et al.: VHDL – Feature Description and Analysis. IEEE Design & Test, April 1986

[2] Gizdoń H., Pawlak A., Wrona W.: VHDL – Ada języków opisu i projektowania sprzętu. Informatyka, nr 10, 1988

[3] Nash J. D., Sanders L. F.: VHDL Critique. IEEE Design & Test, April 1986

[4] Piloty R. et al.: CONLAN Raport, Springer-Verlag, 1983

[5] Roumeliotis M., Armstrong J. R.: HDL Modeling for Process Oriented Simulation. Computer Hardware Description Languages and Their Applications, IFIP, 1987

[6] Shahdad M. et al.: VHSIC Hardware Description Language. IEEE Computer. February 1985

[7] VHDL Language Reference Manual, Version 7.2. Report IR-MD-045-2, 1 August 1985

[8] VHDL Users's Manual. Vol. 1 – Tutorial, Report IR-MD-065-1. 1 August 1985.

PIOTR FUGLEWICZ
ROMAN TOPCZYŃSKI
CNPSS MERASTER
Katowice

DAGGER – program utrzymania plików

W artykule opisano program DAGGER będący przedstawicielem nowej generacji oprogramowania dla komputerów rodziny Mera-60/660 i systemów operacyjnych RT-11, RT-60 lub MRT-60.

Program stanowi rodzaj powłoki (ang. *shell*), eliminującej konieczność korzystania w tradycyjnej formie z większości poleceń systemowych. Jego filozofia jest wzorowana na filozofii dwóch programów dla komputerów IBM PC: NORTON Petera Nortona oraz XTREE firmy Executive Systems.

Program uwzględnia w swoim działaniu – wzorowane częściowo na strukturach stosowanych w systemie Unix – dyski logiczne, wprowadzone dopiero w piątej wersji systemów RT-11 i RT-60 oraz w systemach RAFOS II, MRT-60 i sieci SN-60. Umożliwia szybki dostęp do plików znajdujących się wewnątrz dysków logicznych, co jest dużym ułatwieniem w pracy, zwłaszcza pod kontrolą systemu MRT-60, z urządzeniami dyskowymi o dużej pojemności, zawierającymi wiele dysków logicznych.

GENEZA PROGRAMU

Sposób korzystania z informacji zapisanych na dyskach w systemie RT-60 i zarządzania nimi jest w świetle współcześnie używanych metod (zwłaszcza na komputerach osobistych), wyraźnie przestarzały. Jeśli na przykład użytkownik chce usunąć z dysku większą liczbę plików tekstowych, to musi przy użyciu polecenia **DIR** uzyskać informację o zawartości nośnika, a następnie wielokrotnie używając polecenia **TYPE** sprawdzać zawartość poszczególnych plików, każdorazowo podejmując decyzję, czy usuwać plik poleceniem **DEL**, czy nie. Ponieważ po użyciu polecenia **TYPE** z ekranu ginie informacja o zawartości nośnika, należy więc ponownie użyć polecenia **DIR**, i tak aż do skutku.

Innym problemem w wielodostępnej wersji systemu operacyjnego – MRT-60 – jest podział nośnika na pewną liczbę dysków logicznych, umożliwiających grupowanie plików jednego użytkownika. Często jest potrzebne szybkie przełączanie między dyskami logicznymi, w celu uzyskania dostępu do plików.

Celem powstania programu DAGGER było dostarczenie programiście zarządzającemu systemem nowoczesnego narzędzia, zwiększającego efektywność i komfort pracy.

Program DAGGER jest programem usługowym, umożliwiającym wykonywanie wielu poleceń systemu RT-60 i jego pochodnych, bez konieczności korzystania w tradycyjnej formie z większości poleceń systemowych (**DIR**, **COPY**, **DUMP**, **PATCH**, **TYPE**, **PRINT**, **RUN**). Ponadto program dostarcza użytkownikowi nowych, niedostępnych dotychczas możliwości, pozwalając efektywnie zarządzać plikami znajdującymi się na urządzeniach dyskowych pracujących w systemie. Jedną z nich jest możliwość analizy i obsługi pseudodrzewiastej struktury katalogów i dysków logicznych. Współpraca z użytkownikiem odbywa się w trybie pełnoekranowej wizualizacji danych przy pełnym wykorzystaniu możliwości klawiatury do sterowania pracą programu.

Program DAGGER został napisany prawie w całości w języku C. Jedynie trzy krótkie podprogramy, wykorzystujące żądania programowe, napisano w języku makroasemblera.

FUNKCJE PROGRAMU

Działanie programu DAGGER polega na odczytywaniu i modyfikowaniu danych zapisanych na urządzeniach o dostępie bezpośrednim. Dane wejściowe dla programu DAGGER stanowi struktura urządzenia dyskowego. Każdy dysk można umownie podzielić na dwie części. Część pierwsza (obejmująca początkowe bloki fizyczne urządzenia) zawiera katalog całego urządzenia. Pozostała część jest przeznaczona do przechowywania danych.

W systemach RT-60 v05, MRT-60 oraz w sieci SN-60 istnieje możliwość zagnieżdżania struktury urządzeń dyskowych, tzn. dowolny obszar danych może być dyskiem logicznym. Poziom zagnieżdżania dysków nie jest ograniczony.

DAGGER umożliwia dostęp do informacji na trzech poziomach; z poziomami tymi są związane trzy tryby pracy:
- poziom urządzenia (tryb Volume),
- poziom katalogu (tryb Directory),
- poziom pliku (tryb File).

Na poziomie urządzenia mogą być realizowane następujące funkcje:
– kopiowanie całego urządzenia na inne,
– odszukanie w obrębie urządzenia podanego wzorca,
– dostęp do dowolnego bloku urządzenia z możliwością zmiany jego zawartości,
– zmiana nazwy oraz identyfikatora urządzenia,
– wydruk pseudodrzewiastej struktury urządzenia z nazwami plików znajdujących się w poszczególnych dyskach logicznych.

Na poziomie katalogu można wykonywać następujące czynności:
– weryfikacja poprawności zapisu informacji w katalogu urządzenia oraz korekta wartości niepoprawnych,
– uzyskanie pełnej informacji o plikach wewnątrz katalogu,
– odszukiwanie podanego wzorca w obrębie permanentnych plików zapisanych w danym katalogu lub w obszarze wolnym,
– tworzenie dysków logicznych w obrębie aktywnego katalogu,
– oznaczanie i anulowanie oznaczeń wszystkich plików wewnątrz urządzenia logicznego,
– kopiowanie, usuwanie, przemianowanie oznaczonych plików,
– zmiana atrybutów plików i odtwarzanie plików usuniętych.

Na poziomie pliku jest możliwe:
– sprawdzenie, czy dany plik nie zawiera złych bloków,
– dokonywanie zmian w treści pliku,
– oznaczanie i anulowanie oznaczeń pojedynczych plików,
– odszukiwanie podanego wzorca wewnątrz pliku,
– kopiowanie, usuwanie, przemianowanie, wydruk na terminal i drukarkę pojedynczych plików.

Wymienione funkcje obejmują większość operacji wykonywanych podczas pracy pod kontrolą systemu RT, dając równocześnie użytkownikowi większe możliwości niż sam system.

DZIAŁANIE PROGRAMU

W pierwszym etapie pracy programu DAGGER są uzyskiwane i wyprowadzane na konsolę użytkownika następujące informacje:
– nazwa urządzenia,

– obraz pseudodrzewiastej struktury urządzenia,
– wykaz plików zwykłych poziomu zerowego,
– informacje statystyczne.

Ekran konsoli jest sformatowany w postaci dwóch okien (zwanych umownie górnym i dolnym) oraz kilku wierszy pomocniczych. Obydwa okna są oddzielone jednym wierszem, wyświetlanym w negatywie, symbolizującym tryb Directory.

Pierwszą informacją, która pojawia się u góry ekranu, jest nazwa urządzenia podana przez użytkownika. Następnie kolejno są wyświetlane nazwy rozpoznawanych plików zwykłych oraz dysków logicznych. Z nazw dysków logicznych jest budowany i wyświetlany w górnym oknie obraz drzewiastej struktury urządzenia.

Dolne okno ekranu jest zapełniane nazwami wewnętrznych plików poziomu zerowego. Wykaz plików zwykłych urządzenia może być częściowy (dolne okno mieści maksymalnie 25 nazw plików).

Podczas analizy struktury danego urządzenia w programie są obliczane liczby:
– wewnętrznych plików zwykłych poziomu zerowego,
– wewnętrznych dysków logicznych poziomu zerowego,
– bloków zajętych przez pliki zwykłe,
– bloków zajętych przez dyski logiczne,
– bloków wolnych.

Informacje te są wyświetlane w dwóch pierwszych wierszach ekranu.

```
+-------------------+-----------------------------------------+
!Device/Disk: DR2:     37 disks (30199 blocks)      2310 free blocks!
!                      79 files ( 7423 blocks)                    !
!DR2: *                                                           !
!!___IWONAS.DSK                                                   !
!!___WA.DSK                                                       !
!!___SRTV20.DSK                                                   !
!!___UNIX.DSK                                                     !
!   !___FORTRA.DSK                                                !
!!___MJDISC.DSK                                                   !
!!___ERRLOG.SYS                                                   !
!!___SROMEK.DSK                                                   !
!!___EMBSC.DSK                                                    !
!!___BIER.DSK                                                     !
!!___PDIR.DSK                                                     !
!   !___T0031.DSK                                                 !
! D I R E C T O R Y                                               !
!MASU15.BAK    NCAL.PAS    MASU15.DOC    MASU15.DFR    NCAL.OBJ   !
!LCE7E.MAC     LC7E.MAC    NCIF.COM      NCAL.MAC      DATIME.COM !
!NCAL.BAK      C.COM       DATTIM.COM    A.SYS         FORMA.SAV  !
!ACCESS.SYS    MJ.COM      STAR02.SYS    STAR03.SYS    STARTX.COM !
!FGL.SYS       KORN.COM    INSTAL.COM    FORTH1.DOC    UNIXDC.COM !
!                                                                 !
!VOLUME   Analyse   Copy   Dump   Init   Label   Print   Rename   View!
!COMMANDS   everY   Hide   New-disk   Order   Query   Show        Exit!
+-------------------+-----------------------------------------+
```

Przykładowy obraz ekranu uzyskanego w pierwszym etapie pracy programu DAGGER

Przykładowy obraz ekranu uzyskanego w pierwszym etapie pracy programu DAGGER przedstawiono na wydruku. W wierszach 23 i 24 są wyświetlone nazwy poleceń dostępnych w bieżącym trybie pracy programu.

Drugi etap pracy programu obejmuje tryby Volume, Directory oraz File. Po rozpoznaniu struktury i wyświetleniu drzewa dysków logicznych oraz wykazu plików urządzenia programu przechodzi do pracy w trybie Volume. Bieżący dysk logiczny (na początku pracy jest to urządzenie poziomu zerowego) jest sygnalizowany wyświetleniem jego nazwy w negatywie lub – jeśli monitor ekranowy nie umożliwia takiego wyświetlania – znakiem gwiazdki po prawej stronie tej nazwy. Przełączania między dyskami logicznymi dokonuje się z klawiatury, za pomocą klawiszy oznaczonych strzałkami.

Program rozpoznaje strukturę urządzenia, zmieniając zawartość dolnego okna (wykaz plików zwykłych) oraz informacje statystyczne. Obraz drzewa dysków logicznych pozostaje niezmieniony, chyba że jest wymagane przesunięcie wewnątrz górnego okna ekranu, na przykład wówczas, gdy nazwa żądanego dysku logicznego nie była dotychczas wyświetlana (przejście z pierwszego aktualnie wyświetlonego dysku do poprzedniego lub z ostatniego wyświetlonego do następnego).

Przechodzenie do kolejnych poziomów pracy odbywa się przez naciskanie klawisza ⟨CR⟩, według następującego schematu:

Tabela 1. Polecenia wspólne

Polecenie	Opis
every	Przejście do trybu File z dostępem do wszystkich plików we wszystkich dyskach logicznych.
Hide	Usunięcie z maski przeszukiwań plików o wieloznacznej nazwie (ang. *wild card*) lub danym atrybucie.
New Disk	Zmiana urządzenia (lub dysku logicznego), na którym działa program. Odpowiedź ⟨CR⟩ na pytanie o nazwę nowego urządzenia powoduje przejście do trybu Volume aktualnego urządzenia. Polecenie może być stosowane w celu przejścia do trybu Volume z trybu File lub Directory.
Order	Zmiana kolejności wyświetlania na ekranie nazw plików dla wszystkich trybów pracy oraz polecenia Every. Pliki mogą być porządkowane według: nazwy – **Name**, rozszerzenia – **Ext**, wielkości – **Size**, daty – **Date**, atrybutów – **Atribs**. Porządek sortowania podaje się przez użycie jednej z liter **N, E, S, D, A**, w odpowiedzi na pytanie programu.
Query	Zmiana poziomu ochrony. Poziom ochrony jest logiczną sumą pięciu składników odpowiadających częściowo atrybutom plików: **Bad** – określenie, czy operacje Copy, Rename, Delete mają być wykonywane na plikach o rozszerzeniu BAD; **Deleted** – określenie, czy operacje Copy, Rename, Delete mają być wykonywane na plikach usuniętych z nośnika, dla których zachował się wpis w katalogu; **Protected** – określenie, czy operacje Copy, Rename, Delete mają być wykonywane na plikach z ustawionym w katalogu bitem ochrony; **Replacement** – określenie, czy operacja Copy lub Rename ma być wykonana, jeśli w jej wyniku uległby zniszczeniu plik istniejący w katalogu; **System** – określenie, czy operacje mają być wykonywane na plikach z rozszerzeniem SYS. Po użyciu polecenia wyświetlane są aktualne wartości powyższych parametrów. Użycie jednej z liter **B, D, P, R, S**, powoduje przełączenie stanu z aktywnego na nieaktywny i odwrotnie. Użycie klawisza ⟨CR⟩ powoduje powrót do poziomu pracy, aktualnego w momencie wywołania polecenia. W stanie początkowym wartości wszystkich parametrów są ustawione na stan nieaktywny. Stan nieaktywny oznacza, że pliki z danym atrybutem nie są brane pod uwagę przy wykonywaniu operacji.
Show	W połączeniu z poleceniem Hide ustalenie maski przeszukiwania plików, złożonej z trzech elementów: – nazw plików, – atrybutów, – zakresu dat powstania pliku. Nazwa jest ustalana w sposób wieloznaczny (ang. *wild card*), tak jak w systemie RT-60. Atrybuty pliku mogą mieć następujące wartości: **Bad** – plik o rozszerzeniu BAD; **Deleted** – plik usunięty, tzn. taki, dla którego w katalogu pozostała nazwa i niezerowa długość oraz ustawiony odpowiednio bit, pozostałe obszary są zaliczane do obszaru wolnego na dysku; **Protected** – plik z ustawionym w katalogu bitem ochrony; **System** – plik rozszerzeniem SYS. Po użyciu dyrektywy na ekranie ukazują się trzy pola: **Filename: *.*** **Attributes: Bad ON Deleted ON System ON Protected ON** **Date From: To:** W polu Filename można podać dowolną nazwę wieloznaczną. W polu Attributes użycie liter **B, D, S, P**, powoduje przełączenie wartości atrybutu z aktywnego na nieaktywny i odwrotnie. W polu Date użycie liter **F, T**, pozwala na wprowadzenie daty. Użycie w dowolnym momencie znaku ⟨CR⟩ powoduje powrót do aktualnego poziomu pracy. Analogiczna plansza występuje dla polecenia Hide. Przy przeszukiwaniu najpierw sprawdzane są warunki ustawiane poleceniem Show, a potem Hide.

tryb Volume + ⟨CR⟩→ tryb Directory
tryb Directory + ⟨CR⟩→ tryb File
tryb File + ⟨CR⟩→ tryb File(*)
tryb File(*) + ⟨CR⟩→ tryb Volume

przy czym tryb File(*) oznacza tryb File z wykazem plików na całym ekranie.

Użycie klawisza ⟨CR⟩ powoduje przejście do trybu Directory dla dysku logicznego oznaczonego gwiazdką. Praca programu w tym trybie polega na zarządzaniu plikami umieszczonymi w katalogu głównym bieżącego urządzenia. Możliwe jest między

innymi sprawdzenie i rekonstruowanie struktury katalogu, tworzenie nowego (wewnętrznego) dysku logicznego itp. Polecenia trybu Directory (kopiowanie, kasowanie, zmiana nazw plików itp.) działają na ogół na grupie plików bieżącego urządzenia.

Tabela 2. Polecenia poziomu urządzenia

Polecenie	Opis
Analyse	Odpowiednik polecenia systemowego DIR/BAD dla urządzenia wskazanego przez kursor.
Copy	Odpowiednik polecenia systemowego COP/DEV ⟨dostępne urządzenie⟩ ⟨podane urządzenie⟩
Dump	Wyprowadzenie na drukarkę podanego bloku (bloków) urządzenia w postaci binarnej. Odpowiada poleceniu systemowemu. DUMP/START:xxx/END:xxx dla urządzenia wskazywanego przez kursor (xxx – numery bloków)
Init	Odpowiednik polecenia systemowego INI/BAD dla urządzenia wskazywanego przez kursor.
Label	Zmiana identyfikatora urządzenia. Odpowiada poleceniu INI/VOL: ONLY systemu RT-60.
Print	Wyprowadzenie na drukarkę lub do wyspecyfikowanego pliku graficznego obrazu struktury urządzenia. Opcjonalnie może być dołączony wykaz plików wewnętrznych każdego z dysków logicznych.
Rename	Zmiana nazwy bieżącego urządzenia na nazwę podaną przez użytkownika.
View	Wyświetlenie na terminalu zawartości bloku o podanym numerze w postaci szesnastkowej – lewa kolumna oraz w kodzie ASCII – prawa kolumna. Można dokonywać zmian zawartości bloku wprowadzając nowe wartości w dowolnej kolumnie. Użycie strzałek umożliwia przesuwanie aktywnego kursora wewnątrz ekranu. Naciśnięcie klawisza GOLD oraz strzałki powoduje przejście do bloku poprzedniego lub następnego. Użytkownik może anulować wprowadzone przez siebie zmiany. Dodatkowo polecenie pozwala odnaleźć podany wzorzec szesnastkowo lub w kodzie ASCII w obrębie całego urządzenia.

Użycie klawisza ⟨CR⟩ w trakcie oczekiwania programu na żądanie wykonania polecenia trybu Directory powoduje przejście do trybu File. Bieżącym plikiem staje się pierwszy plik z dolnego okna ekranu. Przechodzenie do sąsiednich plików odbywa się za pomocą klawiszy oznaczonych strzałkami. W razie potrzeby jest wykonywane przesunięcie wewnątrz dolnego okna. Polecenia trybu File dotyczą pojedynczych plików.

Tabela 3. Polecenia poziomu katalogu

Polecenie	Opis
Analyse	Sprawdzenie i rekonstrukcja struktury katalogu urządzenia. Wyświetlany jest katalog urządzenia zawierający wszystkie nazwy plików wewnętrznych oraz obszary puste. Nazwa pliku jest uzupełniona atrybutami pliku. Atrybuty pliku i wszystkie pozostałe informacje, dotyczące pliku, a zapisane w katalogu, mogą być zmieniane. W szczególności plik usunięty może zostać odtworzony. W wypadku, gdyby jego odtworzenie spowodowało usunięcie istniejącego pliku, program pyta o nazwę.
Copy	Kopiowanie do wyspecyfikowanego urządzenia plików oznaczonych za pomocą polecenia Tag. W zależności od poziomu ochrony kopiowanie dotyczy plików określonych tym poziomem.
Delete	Usuwanie zaznaczonych plików, w zależności od poziomu ochrony. Usunięcie każdego pliku wymaga potwierdzenia użytkownika.
Make	Tworzenie nowego dysku logicznego (i zainicjowanie go) wewnątrz aktualnego katalogu.
Print	Wyświetlenie katalogu urządzenia wraz z dodatkowymi informacjami o plikach, niedostępnymi w RT-60. Nazwy plików są wyświetlane w kolejności określonej przez polecenie Order. Katalog urządzenia może być wyprowadzony na drukarkę lub do wskazanego pliku.
Rename	Przemianowanie plików oznaczonych za pomocą polecenia Tag.
Tag	Oznaczenie wszystkich plików bieżącego urządzenia.
Untag	Anulowanie oznaczenia wszystkich plików bieżącego urządzenia.
View	Wyświetlenie segmentów katalogu w kolejności logicznej. Informacje zapisane w tych blokach mogą być modyfikowane analogicznie jak w trybie Volume lub File. Dodatkowo można wyszukiwać podane wzorce w obrębie plików zapisanych w katalogu (podkatalogi nie są traktowane jako pliki) lub w obszarze wolnym.

OPIS POLECEŃ

Część poleceń określających aktualną masę przeszukiwania, kolejność wyświetlania plików na ekranie, nazwę nowego urządzenia itp. jest aktywna i ma jednolite znaczenie, niezależnie od poziomu pracy programu. Przedstawiono je w tabeli 1. Zestaw tych poleceń uzupełnia polecenie **Exit**, które powoduje zakończenie pracy programu i powrót do systemu operacyjnego.

Na poziomie urządzenia (tryb Volume) można używać poleceń zestawionych w tabeli 2. Przejście programu z poziomu urządzenia do poziomu katalogu następuje przez użycie znaku ⟨CR⟩. Kursor zostaje przemieszczony do wiersza ekranu znajdującego się między górnym a dolnym oknem.

Na poziomie katalogu są dostępne polecenia przedstawione w tabeli 3.

Do trybu File przechodzi się z trybu Directory, po użyciu klawisza ⟨CR⟩ lub dyrektywy Every. Na ekranie jest wyświetlany, zgodny z maską przeszukiwania, wykaz plików otwartego katalogu – po przejściu z trybu Directory – lub plików na całym dysku – po użyciu dyrektywy Every. Wciśnięcie klawisza ⟨Cr⟩ powoduje udostępnienie całego ekranu na wykaz plików (bez podziału na okna).

Tabela 4. Polecenia trybu File

Polecenie	Opis
Analyse	Sprawdzenie, czy plik nie zawiera błędnych bloków, których nie można odczytać. Odpowiada poleceniu systemowemu COPYplik NL:
Copy	Skopiowanie pojedynczego pliku wskazanego przez kursor. Operacja jest wykonywana lub nie, w zależności od poziomu ochrony.
Delete	Usunięcie pliku wskazanego przez kursor (uwagi jak w poprzednim punkcie).
Print	Wyprowadzanie pliku wskazanego przez kursor na drukarkę systemową w postaci źródłowej lub binarnej.
Rename	Przemianowanie pliku, wskazanego przez kursor, zgodnie z nazwą podaną przez użytkownika, jeśli pozwala na to poziom ochrony.
Tag	Oznaczenie pliku, wskazanego przez kursor.
Untag	Anulowanie oznaczenia pliku wskazanego przez kursor.
View	Wyświetlenie na ekranie zawartości pierwszego bloku wskazanego pliku. Użytkownik ma możliwość wprowadzenia zmian w pliku. Przemieszczanie aktywnego kursora, zmiana bloku, anulowanie poprawek odbywa się tak samo, jak dla analogicznego polecenia trybu Volume.
eXecute	Uruchomienie programu wskazanego przez kursor.

Kolejne wciśnięcie ⟨CR⟩ powoduje powrót do trybu Volume (ekran zawiera dwa okna, a kursor wskazuje urządzenie w górnym oknie). Polecenia trybu File dotyczą pojedynczych plików. W trybie tym można używać poleceń przedstawionych w tabeli 4.

* * *

Wzbogacanie starzejących się moralnie systemów operacyjnych o nowoczesne środki operatorskie jest ostatnio powszechne, żeby wspomnieć tylko Warsztat Pracy Programisty, opracowany przez MERA-SYSTEM dla maszyn jednolitego systemu, czy CT Commander Spółdzielni „Comtech" – dla systemu RSX. Również jury 3. Ogólnopolskich Targów Oprogramowania SOFTARG '88 podzieliło pogląd o sensowności tworzenia takich programów, przyznając dla DAGGERA nagrodę drugiego stopnia w kategorii oprogramowania systemowego.

Przypominamy, że w 1989 roku
cena INFORMATYKI
będzie wynosić
300 zł za egzemplarz
60 zł cena ulgowa

WOJCIECH CELLARY
JERZY KRĘGLEWSKI
Instytut Automatyki
Politechnika Poznańska

RUTA MAĆKOWIAK
Instytut Komputerowych Systemów
Automatyki i Pomiarów
Wrocław

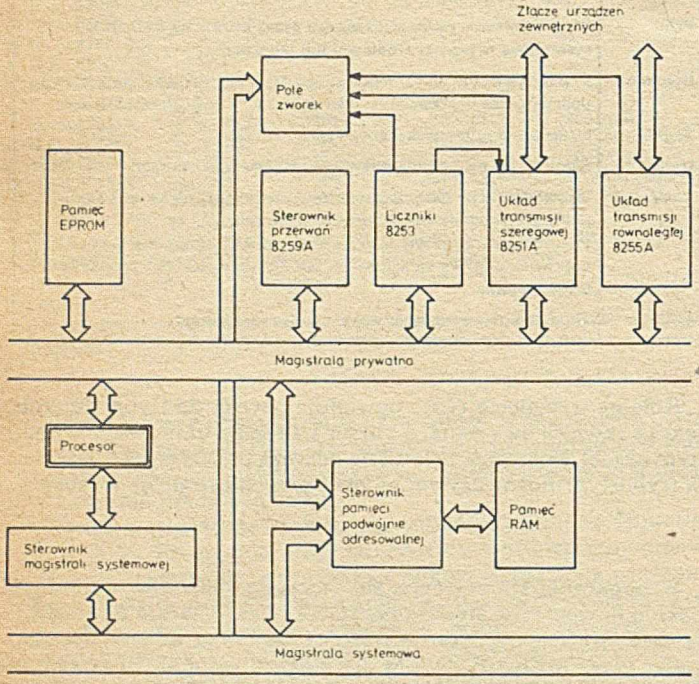
System komputerowy Elwro 800 (2)

W drugiej części artykułu przedstawiono opis poszczególnych modułów funkcjonalnych systemu.

MODUŁ MIKROKOMPUTERA SZESNASTOBITOWEGO M--M16

Moduł M-M16 jest pełnym funkcjonalnie mikrokomputerem szesnastobitowym wykorzystującym mikroprocesor 8086 i koprocesor 8087. Moduł ten zawiera własną pamięć operacyjną typu EPROM i RAM, podstawowe układy wejścia- wyjścia, sterownik przerwań oraz układy sprzęgające z magistralą systemową (rys. 1). W module tym wykorzystywane są cztery magistrale – lokalna, prywatna, systemowa oraz pamięci RAM.

Mikroprocesor 8086 steruje bezpośrednio magistralą lokalną, do której dołączony jest koprocesor arytmetyczny 8087 oraz układy sterowników magistrali prywatnej (tzn. dostępnej wyłącznie w danym module) i systemowej oraz związane z nimi rejestry i wzmacniacze buforujące. Mikroprocesor 8086, koprocesor 8087 i układ zegarowy 8284A tworzą łącznie tzw. procesor modułu.



Rys. 1. Architektura modułu mikrokomputera obliczeniowego

Magistrala prywatna służy do komunikacji pomiędzy procesorem a zasobami prywatnymi modułu, którymi są:
- pamięć operacyjna EPROM,
- układ programowalnych liczników 8253,
- układ programowalnego sterownika transmisji szeregowej 8251,
- układ programowalnego sterownika transmisji równoległej 8255,
- układ programowalnego sterownika przerwań 8259A,
- część pamięci operacyjnej RAM.

Sterownik magistrali prywatnej wykorzystuje układ 8288 oraz rejestry 8282 i dwukierunkowe wzmacniacze 8286.

Maksymalna pojemność prywatnej pamięci operacyjnej EPROM przy wykorzystaniu układów 27512 wynosi 384 KB. Przewidziano też możliwość wykorzystania układów 27256, 27128, 2764, 2732 i 2716, co daje odpowiednio pojemności 192 KB, 96 KB, 48 KB, 24 KB i 12 KB.

Układ programowalnych liczników 8253 zawiera trzy niezależne liczniki, z których jeden połączony jest na stałe z układem sterownika transmisji szeregowej 8251A, a pozostałe dwa mogą być wykorzystane w sposób wybrany przez użytkownika. Układ sterownika transmisji szeregowej 8251A w połączeniu z układem 8253 ma programowo wybieraną szybkość transmisji. Dane wejściowe i wyjściowe oraz sygnały sterowania modemu są doprowadzone poprzez układy sprzęgające dla standardu RS 232 do złącza szufladowego na płycie czołowej modułu. Złącze to jest najczęściej wykorzystywane do dołączenia konsoli operatorskiej.

Układ programowalnego sterownika transmisji równoległej 8255 jest połączony przez układy buforujące z drugim złączem szufladowym płyty czołowej modułu. Złącze to może być wykorzystane na przykład do dołączenia drukarki.

Układ programowalnego sterownika przerwań 8259A jest przeznaczony do przyjmowania przerwań z różnych źródeł. Wybór źródła jest dokonywany za pomocą układu zworek. Umożliwia to odbieranie przerwań od opisanych powyżej układów prywatnych oraz z magistrali systemowej. Należy zaznaczyć, że przerwania przychodzące z magistrali systemowej mogą pochodzić od innych sterowników przerwań typu 8259A, pracujących jako sterowniki podległe. Umożliwia to rozbudowę systemu przerwań przyjmowanych za pośrednictwem omawianego sterownika do 64 niezależnych przerwań wektorowych.

Magistrala systemowa zapewnia modułowi M-M16 możliwość komunikowania się z innymi modułami, umożliwiając założenia wielomodułowego systemu jednoprocesorowego oraz wieloprocesorowego. Sterownik magistrali jest zbudowany z układu 8288, rejestrów 8283, wzmacniaczy 8287 oraz układu arbitrażu 8289. W układzie sterowania magistralą systemową zastosowano również dodatkowy układ rozszerzający możliwość adresowania pamięci operacyjnej przez mikroprocesor 8086 z 1 MB do 16 MB.

W module M-M16 pamięć operacyjna RAM ma pojemność 128 KB. Jest to pamięć podwójnie adresowana, która przez magistralę pamięci i sterownik jest połączona zarówno z magistralą prywatną, jako pamięć prywatna, jak i systemową. Dzięki temu jest widziana przez procesor własnego modułu, a jednocześnie jako pamięć systemowa (wspólna) jest dostępna dla procesorów innych modułów. Sterownik magistrali pamięci zawiera układ arbitrażu uniemożliwiający jednoczesny dostęp do niej. Sterownik ten pozwala jednocześnie ograniczyć dostęp z magistrali systemowej jedynie do wybranych obszarów pamięci. Ograniczenie dostępu z innych modułów do pamięci prywatnej zwiększa niezawodność systemu. Inną zaletą pamięci podwójnie adresowanej jest możliwość bezpośredniej transmisji danych do (lub z) pamięci prywatnej RAM modułu mikrokomputera obliczeniowego, z modułu specjalizowanego mikrokomputera sterującego, co zapewnia istotne podwyższenie efektywności systemu. Ponadto możliwe jest budowanie systemów wielomikroprocesorowych złożonych wyłącznie z modułów mikrokomputerów obli-

SPIS TREŚCI ROCZNIKA

INFORMATYKA 1988

ARTYKUŁY PROBLEMOWE

	nr	s.
ABRAMOWICZ WITOLD – Nowy system pamiętania w systemach informacyjnych	7	17
APPELRATH HANS-JÖRGEN – System wyszukiwawczy dla płyty wizyjnej (tłum. Witold Abramowicz)	7	1
BARTYŚ MICHAŁ – Procedury INTR i MS DOS w MS-Pascalu	5	12
BAUMBACH HANS-DIETER – Stan i tendencje rozwoju technologii programowania w Kombinacie Robotron (tłum. Małgorzata Różańska)	1	4
BIELECKI JA[illegible] – Język C		
Wskazania, adresy, konwersje	1	24
Skojarzenia parametrów z argumentami	2	16
Coraz bliżej normy – część 1	3	16
Coraz bliżej normy – część 2	4	19
Zbiory nagłówkowe, prototypy, deklarowanie funkcji	5	17
Preprocesor i makrodefinicje	6	19
Włączanie warunkowe	7	19
BIELECKI JAN – Turbo C		
Zmienne ustalone, zmienne nietrwałe, punkty charakterystyczne	8	17
Modele pamięci, adresowanie, wskazania adresowe	10	17
Przerwania, sygnały, powroty	11-12	35
BIESIADA HENRYK, GRZYWOCZ JERZY – Komputerowe przetwarzanie wyników badań medycznych	5	19
BŁASZCZAK SŁAWOMIR – Język Occam		
Część 1	7	4
Część 2	8	12
CABAN DARIUSZ – Transputer – narzędzie przetwarzania równoległego	6	10
CELLARY WOJCIECH, KRĘGLEWSKI JERZY, MAĆKOWIAK RUTA – System mikrokomputerowy Elwro 800		
Część 1	10	12
Część 2	11-12	26
CHYŁA ROBERT, MUSIAŁ KRZYSZTOF – Metody kompresji komunikatów	1	26
CIECHOMSKA HALINA, WÓJCIEKIAN TERESA, ZABOROWSKA EWA – PROMIS – system zarządzania bazą danych		
Część 1	5	1
Część 2	6	16
CZACHÓRSKI TADEUSZ, KOWALÓWKA MAREK, SZCZERBIŃSKI ZDZISŁAW, WILK ANDRZEJ – AMOK – mikrokomputerowy system doradczy projektanta i analityka systemów komputerowych	11-12	19
DZIK KRZYSZTOF – Mazovia 1016 – konfiguracja sprzętowa	5	15
FOERSTERLING FRANK – Wymagania stawiane bazom danych (tłum. A. Radomski)		
Część 1	8	5
Część 2	9	12
FRANCZYK JERZY – Komputerowe wspomaganie projektowania układów cyfrowych w języku Modlan	9	23
FRYŹLEWICZ ZBIGNIEW – Mikrokomputerowy analizator-tester protokołów	2	17
FUGLEWICZ PIOTR, KORNIAK TADEUSZ – Projektowanie i implementacja wielodostępnego systemu operacyjnego na przykładzie systemu MASTER	7	7
FUGLEWICZ PIOTR, TOPCZYŃSKI ROMAN – DAGGER – program utrzymania plików	11-12	23
GĄSIOREK WŁADYSŁAW – Pewne problemy konstrukcji i oprogramowania systemowego dla Mery 300	1	11
GIZDOŃ HENRYK, PAWLAK ADAM, WRONA WŁODZIMIERZ – VHDL – Ada języków opisu i projektowania sprzętu	10	8
GIZDOŃ HENRYK, PAWLAK ADAM, WRONA WŁODZIMIERZ – Język opisu sprzętu VHDL – podstawowe mechanizmy		
Część 1	11-12	15
GRISWOLD RALPH E., GRISWOLD MADGE T. – Języki wysokiego poziomu do przetwarzania napisów – COMIT, SNOBOL4 i Icon (tłum. Zdzisław Płoski)		
Część 1	4	1
Część 2	5	4
Część 3	6	4
HUZAR ANDRZEJ, KALIŚ ANDRZEJ – Sieć komputerowa Jednolitego Systemu SKJS2, wersja 1	6	14
ISZKOWSKI WACŁAW – Nasze wspólne problemy	1	1
JAWORSKI ROBERT, PŁOWIEC KRZYSZTOF – Monitor ekranowy Mazovii 1016	2	13
KABSCH MICHAŁ – Efektywność systemu dBASE III	10	15
KALETA GRZEGORZ, OWCZARCZYK JACEK		

	nr	s.
– Wieloprocesorowa realizacja operacji przetwarzania obrazów	8	8
KIRPLUK MICHAŁ, SOBOLEWSKI PIOTR – Implementacja dedukcyjnej bazy danych HOLMES		
Część 1	2	20
Część 2	3	12
KRECZMAR ANTONI – Języki obiektowe		
Część 1	1	2
Część 2	2	1
KRĘPSKI ARTUR – Język Smalltalk-80		
Część 1	3	4
Część 2	4	16
KRÓLIKOWSKI ZBYSZKO – Mikrokomputerowe systemy zarządzania bazami danych	4	6
KRUSZEWSKA DANUTA – System operacyjny IPIX	9	5
KUC MARIUSZ (oprac.) – Jeszcze raz o Turbo Pascalu	6	21
KUC MARIUSZ (oprac.) – Użycie funkcji DOS-a w Turbo Pascalu	8	19
LAMPORT LESLIE – Uproszczone specyfikowanie systemów współbieżnych (tłum. Janusz Zalewski)		
Część 1	10	1
Część 2	11-12	7
ŁUBIŃSKA KATARZYNA – System dydaktyczny do nauki języka ADA/SM	2	8
MICHALSKI RYSZARD S. – O naturze uczenia się – problemy i kierunki badawcze		
Część 1	2	4
Część 2	3	8
MOKRZYCKI WOJCIECH – Techniki obrazowania informacji o trójwymiarowych zdyskretyzowanych scenach	7	13
MOKRZYCKI WOJCIECH – Przegląd technik grafiki komputerowej		
Część 1	9	1
Część 2	10	25
MYKOWIECKA AGNIESZKA, OSTASZEWSKA ANNA – Rozmowa z komputerem	11-12	11
PAPROCKI ANDRZEJ – ADA/SM – kompilator podzbioru Ady dla komputerów SM-4	1	6
PŁOSKI ZDZISŁAW – Chiwriter – procesor tekstu		
Część 2	1	18
PODRAZA ROMAN – Sterowanie przepływowe		
Część 1. Opis modelu	9	19
Część 2. Implementacja modelu	10	20
POPKO JANUSZ – Jednostka centralna Mazovii 1016	1	9
RYBUS RYSZARD – Nowoczesne programowanie w języku C	9	15
STAWICKI JERZY – Narzędzia tworzenia systemów ekspertowych na mikrokomputerach	4	3
STRUTYŃSKI PIOTR (oprac.) – Przegląd sieci lokalnych dla IBM PC	11-12	37
SYFERT ANNA, RAZNOWIECKI ANDRZEJ (oprac.) – Struktura systemu operacyjnego PC-DOS		
Część 4	1	21
Część 5	2	10
Część 6	3	18
SYSŁO MACIEJ M. – Algorytmy kombinatoryczne i ich efektywność		
Część 1. Problem najkrótszego drzewa rozpinającego	3	1
Część 2. Problem plecakowy	4	13
Część 3. Kolorowanie wierzchołków grafu	5	8
SYSŁO MACIEJ M. – Uwagi o powszechnym kształceniu informatycznym studentów wyższych uczelni	3	23
SYSŁO MACIEJ M. – Maszyny i algorytmy równoległe		
Część 1	10	5
Część 2. Maszyny MIMD i SIMD	11-12	30
TUCHOLSKI ADAM – NETBIOS – zasada działania i sposób użytkowania		
Część 1	11-12	1
TUROWICZ ANDRZEJ – Lokalna sieć komputerowa D-Link		
Część 1	6	7
Część 2	7	21
Część 3	8	15
TURSKI WŁADYSŁAW M. – Kanoniczny krok procesu programowania		
Część 1	8	1
Część 2	9	9
WARDA ANDRZEJ – Pamięci dyskowe Mazovii 1016		
Część 1	3	21
Część 2	4	21
ZABRODZKI JAN – Cyfrowe generowanie obrazów		
Część 1	6	1
Część 2	7	10
ZALEWSKI JANUSZ (oprac.) – Kermit – organizacja implementacja i zastosowanie	9	27
ZIELCZYŃSKI PIOTR – TURBO PROLOG. Porównanie dostępnych wersji PROLOGU	1	14
ZIELIŃSKI KRZYSZTOF, INDULSKA JADWIGA, MAGURA-WITKOWSKI PAWEŁ, WALASEK TOMASZ – Architektura oprogramowania lokalnej sieci komputerowej UMMLAN-2	4	9

Z KRAJU

Mikrokomputery z Domu Handlowego Nauki – **J. D.**	4	27
BALTCOM'87 – **Jarosław Deminet**	4	28
INFOSYSTEM'88 – **Jarosław Deminet**	5	23
Komputery optoelektroniczne – **J. R.**	6	13
Konkurs PTI na prace magisterskie z informatyki – rozstrzygnięty – **Barbara Osuchowska**	8 okł.	III

ZE ŚWIATA

Robotron na LFM'87 – **Władysław Klepacz**	1	13
Systemy personalne IBM PS/2 – nowość czy tylko face-fifting? – oprac. **Wacław Iszkowski**	1	28
W pracowniach IBM – oprac. **Dorota Inkielman**	2	23
Pascal na mikrokomputery IBM PC – oprac. **M. Kuc**	2	24
Projekt komputerów piątej generacji – **Krzysztof Michalik**	2	28
Komputerowe nauczanie Ady – oprac. **Dorota Inkielman**	3	27
Co to jest MAP? – oprac. **Andrzej Raznowiecki**	4	23
Podejście do tworzenia kompilatorów Ady – oprac. **Dorota Inkielman**	4	24
Akademicka sieć superkomputerowa – oprac. **Marek Machura**	4	26
Nowe superkomputery – **M. Machura**	4	29
Kompilator Ady dla mikroprocesora MC800 – oprac. **Dorota Inkielman**	5	24
Systemy ekspertowe – stan obecny i przyszłość – oprac. **M. Machura**	5	26
Esperanto jako język pośredni w tłumaczeniu komputerowym – oprac. **R. Grabowicz**	5	28

	nr	s.
W pogoni za efektywnością komputerów – oprac. **Dorota Inkielman**	6	26
Kompilator Ady dla IBM PC/AT – oprac. **Mariusz Kuc**	6	27
Rodzina superkomputerów ETA10 – **M. M.**	6	28
Bardzo szybka jednostka centralna firmy Inmos – **J.R.**	6	29
Zagrożenie planów opracowania szybkich komputerów w Japonii – oprac. **Jan Ryżko**	6	30
Skrośny system firmy Alsys wspomagania programowania w Adzie dla PC/AT – oprac. **M. Kuc**	7	23
Nowe bloki systemu Multibus II – **Krzysztof Rzymkowski**	7	25
Informatyka w Chinach – oprac. **J. Ryżko**	7	27
Cztery kompilatory Ady na mikrokomputery PC – oprac. **M. Kuc**		
Część 1	8	23
Część 2	9	30
Niektóre pakiety do sterowania dla IBM PC – oprac. **Jerzy Strycharczyk**	8	25
Nowe stanowiska komputerowe dla sztucznej inteligencji – **M. Machura**	8	29
Perspektywy rozwoju układów ASIC – oprac. **Jan Ryżko**	11-12	40
Rynek drukarek komputerowych w 1988 roku – oprac. **Jan Ryżko**	11-12	42
Program bibliotekarza Ady – oprac. **Dorota Inkielman**	11-12	44
Środowisko programowe Ady – oprac. **Dorota Inkielman**	11-12	46
Kto jest kim w IFIP		
Zmarł profesor Tohru Moto-oka	4	26
Ping-Xin Guo	7	27
Herve Gallaire	8	28
Aage Melbye	9	III okł.
Aad van Wijngaarden	10	III okł.
Wolfgang D. Ehrenberger	11-12	okł. 50

TERMINOLOGIA

	nr	s.
Język C – propozycja polskiej terminologii – **Jan Bielecki**		
Część 2	1	30
Czy można lekceważyć niechlujstwo językowe – **W.M.T.**	2	31
Terminologia języka Smalltalk-80 – **Artur Krępski**	3	29
Basic po polsku – **Janusz Zalewski**		
Część 1	4	31
Część 2	8	30
Pliki i zbiory – **Jan Bielecki, Jan Madey**	5	30
System plików – **Wacław Iszkowski**		
Część 1	6	31
Część 2	7	30
Terminologia systemów operacyjnych – **Janusz Zalewski**		
Część 1	9	29
Część 2	10	29
Część 3	11-12	III okł.

RECENZJE

	nr	s.
Nowe czasopismo poświęcone systemom z bazami wiedzy. Knowledge-Based Systems – **M. Machura**	8	29
Wprowadzenie do grafiki komputerowej I.O. Angella – **Wojciech Mokrzycki**	10	24
Data & Knowledge Engineering – **W. Rzeczkowski**	11-12	51

KONFERENCJE

	nr	s.
VI Ogólnopolskie Targi Wynalazków	1	23
Zastosowanie mikrokomputerów w informacji naukowej, technicznej i ekonomicznej	2	7
INFOPROD'88	3	15
MASAD'88	6	25
Mikroprocesory w elektrotechnice	7	18
Sieci komputerowe'89	9	IV okł.
CeBIT'89	10	11
SOFTARG'88	11-12	29
Lokalne sieci komputerowe dla automatyzacji przemysłu	11-12	36
Polioptymalizacja w projektowaniu	11-12	50
Teoretyczne podstawy informatyki. Deklaracja uczestników Międzynarodowej Konferencji FCT'87	11-12	48

NOWE KSIĄŻKI

	nr	s.
Wydawnictwa Naukowo-Techniczne	2	3
Wydawnictwa zagraniczne	4	30
Wydawnictwa zwarte Centralnej Biblioteki Technicznej NOT	4	30
Wydawnictwa zagraniczne	4	15
	4	18
Problematyka baz danych	5	31
Wydawnictwa Naukowo-Techniczne	6	9
Język Ada	7	29
Wydawnictwa Naukowo-Techniczne	10	31
Wydawnictwa nt. NETBIOS-a	11-12	6

Zamieszczona obok tabela zawiera tematyczny wykaz wszystkich artykułów problemowych z lat 1986-1988, wliczając w to artykuły oryginalne, tłumaczenia artykułów oraz opracowania dokonane na podstawie innych artykułów. Artykuły wieloczęściowe uwzględniono tylko raz, podobnie jak „seriale" opisujące języki programowania (np. język C.)

Tematyka	Lata			Suma
	1986	1987	1988	
Języki programowania	12	8	10	30
Systemy operacyjne	4	8	3	15
Inżynieria i techniki programowania	5	3	2	10
Komunikacja i sieci komputerowe	6	2	5	13
Bazy danych	2	7	4	13
Sztuczna inteligencja	2	5	3	10
Grafika komputerowa	–	2	2	4
Algorytmy i metody obliczeniowe	1	1	2	4
Narzędzia podstawowe i pakiety	3	5	2	10
Programy dydaktyczne i nauczanie informatyki	–	3	2	5
Architektury komputerowe	4	1	3	8
Sprzęt (włączając mikroprocesory)	5	6	7	18
Inne	6	1	3	10

NIE DAJEMY RECEPT
SPRZEDAJEMY NARZĘDZIA

ELEKTRONIKA FILM KOMPUTER
Zakład Spółdzielni Pracy UNICUM

ul. Barska 3/20, 02-315 Warszawa
tel. 23-67-57, tlx 816955

OFERUJE USŁUGI W DZIEDZINIE KOMPUTERYZACJI PRZEDSIĘBIORSTW

Podejmuje się kompleksowej obsługi kontrahentów:

- **sprzedaż sprzętu mikrokomputerowego (kompletacja, dostawa, serwis gwarancyjny i pogwarancyjny)**

- **opracowanie oprogramowania użytkowego (wdrożenie, szkolenie personelu)**

Służymy Państwu:

- **doradztwem organizacyjnym**
- **projektowaniem, oprogramowaniem oraz wdrażaniem systemów dedykowanych dla konkretnego użytkownika**
- **opracowaniem unikalnych programów wraz z nadzorem autorskim**

Sprzedajemy profesjonalne narzędzia dla profesjonalistów.

EO/955/88

czeniowych, gdyż nie jest konieczne stosowanie modułu pamięci systemowej do wymiany pomiędzy nimi komunikatów.

MODUŁ MIKROKOMPUTERA OŚMIOBITOWEGO M-M08

Moduł M-M08 jest funkcjonalnie pełnym, jednopłytowym mikrokomputerem ośmiobitowym wykorzystującym mikroprocesor 8080A. Ma on taką samą architekturę wewnętrzną jak omówiony powyżej moduł M-M16. W porównaniu z M-M16 uproszczono w nim organizację magistral, pozostawiając jedynie dwie magistrale – prywatną i systemową. Brak magistrali lokalnej wynika z braku możliwości dołączenia koprocesora. Brak magistrali pamięci RAM zostanie wyjaśniony poniżej.

Mikroprocesor 8080A wraz z układem zegarowym 8224 i sterownikiem 8228 steruje magistralą prywatną, do której dołączono:
- pamięć operacyjną EPROM,
- układ programowalnych liczników 8253,
- układ programowalnego sterownika transmisji szeregowej 8251A,
- układ programowalnego sterownika transmisji równoległej 8255,
- układ programowalnego sterownika przerwań 8259,
- pamięć operacyjną RAM.

Pamięć operacyjna EPROM ma przy zastosowaniu układów 27128 pojemność 32 KB. Istnieje możliwość zastosowania układów 2764, 2732, 2716, co odpowiada pojemnościom pamięci odpowiednio 16 KB, 8 KB i 4 KB.

Rozwiązania prywatnych układów wejścia-wyjścia są analogiczne do rozwiązań przyjętych w module M-M16. Różnica występuje w odniesieniu do programowalnego sterownika przerwań i wynika ze standardu magistrali systemowej AMS. Standard ten dopuszcza stosowanie tylko jednego typu przerwań wektorowych. Sekwencje przyjęcia przerwania (INTA) dla mikroprocesorów 8080A oraz 8086 różnią się i dlatego dla danego systemu konieczny jest wybór typu mikroprocesora, dla którego będą stosowane przerwania wektorowe. W systemie Elwro 800 przyjęto, że hierarchiczne systemy przerwań będą tworzone dla mikroprocesora 8086.

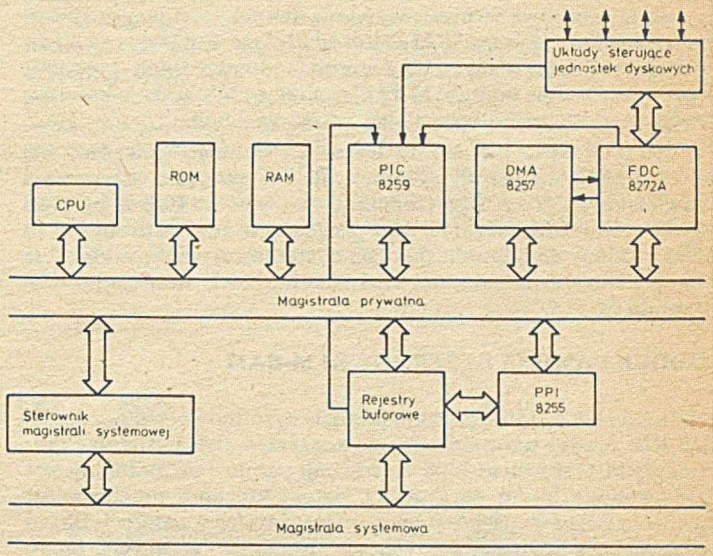
Magistrala systemowa modułu M-M08 jest sterowana przez sterownik zbudowany z układu 8218 i dwukierunkowych wzmacniaczy 8287. Warto podkreślić, że układ arbitrażu 8218 został uzupełniony o dodatkową logikę, dzięki czemu wykorzystuje on w pełni możliwości systemu arbitrażu magistrali AMS i zapewnia większą efektywność pracy mikrokomputera ośmiobitowego przy korzystaniu z magistrali systemowej. W układzie sterowania magistralą systemową zastosowano również układ rozszerzający możliwości adresowania pamięci przez mikroprocesor 8080A z 64 KB do 1 MB, a rejestrów wejścia-wyjścia z 256 B do 64 KB.

Pamięć RAM modułu M-M08 jest również pamięcią podwójnie adresowaną. Brak magistrali pamięci RAM wynika z przyczyn technologicznych i był podyktowany koniecznością ograniczenia liczby elementów na tyle, aby możliwa była realizacja modułu M-M08 na jednej płycie. Jednak brak wydzielonej magistrali pamięci RAM spowodował konieczność wstrzymania pracy mikroprocesora 8080A na czas dostępu do jego prywatnej pamięci RAM innego mikroprocesora od strony magistrali systemowej. Wstrzymanie to obejmuje zazwyczaj okres tylko dwóch cykli zegarowych i dlatego przy poprawnie zaprojektowanym oprogramowaniu spadek efektywności mikrokomputera ośmiobitowego z tej przyczyny jest pomijany.

MODUŁ STEROWANIA DYSKÓW ELASTYCZNYCH M-FLO

Moduł M-FLO jest specjalizowanym mikrokomputerem przeznaczonym do sterowania jednostkami pamięci na dyskach elastycznych oraz wstępnego przetwarzania danych transmitowanych do i z minikomputera obliczeniowego (rys. 2).

Moduł M-FLO jako inteligentny sterownik pamięci dyskowej przyjmuje i interpretuje rozkazy otrzymywane z mikrokomputera obliczeniowego i zwrotnie przekazuje informacje o stanie wykorzystania rozkazów. Oprogramowanie sterujące modułu umożliwia dostosowanie formatu tych rozkazów do wymagań systemu operacyjnego pracującego na mikrokomputerze obliczeniowym. Dotyczy to zarówno sposobu przedstawienia parametrów jak i zakresu funkcji możliwych do realizacji. Dzięki możliwości przetwarzania lokalnego moduł M-FLO może przejąć od mikrokomputera obliczeniowego wszystkie funkcje związane z fizyczną obsługą jednostek pamięci dyskowej, odciążając go w ten sposób od wykonywania tych stosunkowo złożonych i czasochłonnych funkcji. Oprogramowanie sterujące umożliwia ponadto dostosowanie parametrów formatu zapisu danych na dysku do wymagań systemu operacyjnego, pracującego na mikrokomputerze obliczeniowym. Możliwa jest na przykład zmiana wielkości sektora dyskowego na dyskietce, przy zachowaniu niezmienionej wielkości sektora logicznego widzianego przez system operacyjny.



Rys. 2. Architektura modułu sterownika dysków elastycznych M-FLO

Niejednokrotnie w systemach mikrokomputerowych powstaje problem dołączenia różnych jednostek dyskowych i stosowania różnych standardów zapisu na dyskietkach. W module M-FLO układy współpracujące ze sterownikiem FDC umożliwiają dołączenie czterech jednostek pamięci dyskowej dowolnego typu (jednostronne lub dwustronne, pojedynczej lub podwójnej gęstości zapisu 5,25″ i 8″).

Pracą modułu M-FLO steruje mikroprocesor 8080A, który przez magistralę prywatną jest połączony z prywatną pamięcią operacyjną EPROM i RAM oraz programowalnymi układami wejścia-wyjścia, sterownikiem przerwań 8259, układem transmisji równoległej 8255, sterownikiem kanału bezpośredniego dostępu (DMA) 8257 i sterownikiem jednostek pamięci dyskowych (FDC) 8272A. Pamięć EPROM o maksymalnej pojemności 12 KB (przy zastosowaniu układów 2732) zawiera program sterujący pracą modułu. Pamięć RAM o pojemności 2 KB (zastosowano układy 2114) jest przeznaczona do przechowywania zmiennych i stosu programu sterującego oraz wykorzystywana jest jako bufor dla transmisji dyskowych.

Sterownik przerwań 8259 przyjmuje zgłoszenia pięciu różnych przerwań – trzy są związane ze współpracą z magistralą systemową, a pozostałe z zegarem i ze sterownikiem FDC.

Sterownik transmisji równoległej jest wykorzystywany do przyjmowania rozkazów kierowanych do modułu M-FLO z magistrali systemowej oraz do sterowania tą magistralą w trakcie transmisji. Sterownik kanału bezpośredniego dostępu DMA jest wykorzystywany do prowadzenia transmisji danych między sterownikiem jednostek pamięci dyskowych a prywatną pamięcią lub pamięcią systemową modułu M-FLO. Bezpośrednia transmisja danych z dysku do pamięci systemowej (lub w odwrotnym kierunku) jest stosowana wtedy, gdy długość sektora logicznego jest równa długości sektora fizycznego oraz gdy dla

operacji czytania z dysku dopuszczalne jest przesłanie do pamięci informacji niepełnej lub błędnie odczytanej.

W zastosowaniach dopuszczających wyłącznie przesyłanie poprawne informację z dysku transmituje się wstępnie do bufora w pamięci prywatnej i dopiero po stwierdzeniu, że jest poprawna, do pamięci systemowej. W celu efektywnej realizacji transmisji z buforowaniem sektora, sterownik DMA został uzupełniony dodatkowymi układami umożliwiającymi prowadzenie szybkich transmisji typu pamięć-pamięć. Transmisje takie mogą być dokonywane pomiędzy pamięcią prywatną modułu M-FLO, a dowolnym obszarem pamięci systemowej w zakresie 16 MB przestrzeni adresowej.

Moduł M-FLO jest widziany przez mikrokomputery obliczeniowe jako zbiór rejestrów wejścia-wyjścia dostępnych przez magistralę systemową.

Rejestry te służą do przekazywania danych sterujących i informacyjnych do modułu M-FLO lub odwrotnie. Zapis do rejestrów sterujących powoduje uaktywnienie odpowiednich procedur oprogramowania modułu M-FLO realizujących funkcje komunikacji oraz operacji odczytu i zapisu na dyskach.

Moduł M-FLO jes zdolny do samodzielnego ubiegania się o dostęp do magistrali systemowej. Po przejęciu sterowania magistralą może on albo transmitować dane do (lub z) pamięci systemowej, albo wykonywać program w niej zapisany. Ten drugi sposób dostępu do pamięci systemowej daje praktycznie nieograniczone możliwości rozszerzenia cech funkcjonalnych modułu M-FLO.

MODUŁ PAMIĘCI SYSTEMOWEJ M-RAM

Moduł M-RAM jest blokiem pamięci systemowej o pojemności 512 KB. Moduł ten może być wykorzystywany zarówno przez moduły M-M16 jeśli M-M08. Każdy bajt pamięci uzupełniony jest dodatkowym bitem parzystości, dzięki któremu możliwa jest kontrola poprawności danych odczytywanych z pamięci. Układ kontroli poprawności danych w momencie wykrycia błędu sygnalizuje ten fakt generując przerwanie dla modułu mikrokomputera. Ponadto układ kontroli zapamiętuje dokładny adres komórki pamięci, w której wystąpił błąd, umożliwiając w ten sposób szybką lokalizację błędnej danej w pamięci przez oprogramowanie mikrokomputera korzystającego z modułu M-RAM. Układ adresowania pamięci modułu M-RAM umożliwia umieszczenie go jako bloku pamięci w przestrzeni 16 MB, z adresem początkowym będącym wielokrotnością 32 KB.

Układy pamięciowe oraz układy niezbędne do odświeżania pamięci są zasilane z linii zasilania buforowego, dzięki czemu moduł M-RAM może być stosowany w systemach, w których wymagana jest odporność na zanik zasilania w sieci energetycznej. W module zastosowano układy pamięci dynamicznej o pojemności 64 K × 1 i układ sterownika pamięci dynamicznych 8203.

MODUŁ PAMIĘCI SYSTEMOWEJ M-ECC

Moduł M-ECC jest blokiem pamięci systemowej o pojemności 256 KB, przeznaczonym dla systemów o podwyższonej niezawodności. W pamięci tej zastosowano układ kodujący i generujący kod korekcyjny Hamminga, umożliwiający automatyczną korekcję pojedynczych błędów w słowie szesnastobitowym. Kod korekcyjny oraz zasada sterowania elementami pamięciowymi są wzorowane na rozwiązaniach zastosowanych w układzie firmy Intel 8206.

MODUŁ GRAFIKI KOLOROWEJ M-CRT

Moduł M-CRT jest sterownikiem monitora kolorowego umożliwiającym generowanie obrazów alfanumerycznych lub graficznych w szesnastu kolorach. W trybie alfanumerycznym format obrazu (80 znaków × 25 wierszy lub 40 znaków × 25 wierszy) jest wybierany programowo. Dla całego ekranu wybrany jest również zestaw dostępnych kolorów. W każdym zestawie możliwe jest tworzenie obrazów w czterech kolorach z palety szesnastu kolorów.

W trybie graficznym możliwa jest rozdzielczość 640 × 200 punktów z obrazem monochromatycznym lub rozdzielczość 320 × 200 punktów z obrazem kolorowym.

W module zastosowano sterownik scalony firmy Motorola MC6845. Sygnał wyjściowy jest dostosowany do sterowania monitorem z wejściem RGB oraz ma wyjście Compose Video do sterowania monitorem monochromatycznym (np. Neptun 156).

Moduł M-CRT jest kompatybilny z kartą Color Graphics Adapter IBM.

MODUŁ STEROWNIKA MONITORA TELEWIZYJNEGO M-VDO

Moduł M-VDO jest sterownikiem monitora monochromatycznego umożliwiającym generowanie obrazów alfanumerycznych i semigraficznych. W trybie alfanumerycznym możliwe są dwa formaty: 80 znaków × 24 wiersze lub 64 znaki × 24 wiersze. W trybie semigraficznym każde pole znakowe jest podzielone na sześć elementów, tworząc w ten sposób mozaikę o rozdzielczości 160 × 72 lub 128 × 72 elementy. Oba podane tryby pracy mogą być wykorzystywane wspólnie tworząc obrazy semigraficzne z opisami alfanumerycznymi. Do wyróżnienia wybranych fragmentów obrazu dostępne są dwa atrybuty: podwójna jaskrawość oraz obraz negatywowy.

Moduł zawiera również układ 8251A i 8255 służące do dołączenia klawiatury oraz dodatkowego urządzenia z interfejsem IRPR.

MODUŁ KOMUNIKACYJNY M-COM

Moduł M-COM jest sterownikiem komunikacyjnym przeznaczonym do obsługi transmisji z protokołami BSG, SDLC oraz transmisji asynchronicznej. Moduł zawiera układy programowalne 8251A do realizacji prototołu BSC i transmisji asynchronicznej oraz układ 8273 do realizacji protokołu SDLC. Dodatkowo, wykorzystywany jest układ 8255 do sterowania i monitorowania linii dodatkowych łączy transmisyjnych.

Pod względem funkcjonalnym moduł M-COM odpowiada połączonym sterownikom komunikacyjnym mikrokomputera IBM PC.

MODUŁ STEROWNIKA DYSKÓW STAŁYCH WINCHESTER M-WDC

Moduł M-WDC jest specjalizowanym mikrokomputerem przeznaczonym do sterowania jednostkami pamięci na dyskach stałych Winchester oraz wstępnego przetwarzania danych transmitowanych do i z mikrokomputera obliczeniowego. Architektura tego modułu jest zbliżona do architektury modułu M-FLO. W części specyficznej modułu M-WDC, zawierającej układy współpracujące bezpośrednio z jednostkami dyskowymi, wykorzystano układ scalony typu 82062. Do modułu sterownika M-WDC można dołączyć maksymalnie cztery jednostki dyskowe wyposażone w interfejs w standardzie ST506 lub ST412. Każda z jednostek dyskowych może mieć inną organizację nośnika, tj. różną liczbę ścieżek i różną liczbę głowic. Maksymalna liczba ścieżek jednej jednostki dyskowej obsługiwanej przez sterownik wynosi 1024, a maksymalna liczba głowic jest równa 16. Przyjęty standard ST506/ST412 umożliwia dołączenie do modułu M-WDC niemal wszystkich produkowanych obecnie w świecie jednostek dyskowych 3,5-calowych i 5,25-calowych.

MODUŁ STEROWNIKA WIELOKANAŁOWEJ TRANSMISJI SZEREGOWEJ M-V24

Moduł sterownika wielokanałowej transmisji szeregowej w standardzie V-24 jest specjalizowanym mikrokomputerem przeznaczonym do obsługi grupy terminali operatorskich przy pracy mikrokomputerów Elwro 800 w systemie wielodostępnym. Mo-

duł M-V24 umożliwia dołączenie sześciu terminali oraz dwóch drukarek. Terminale operatorskie przy zastosowaniu standardowego łącza RS-232 C mogą być oddalone do 50 m od mikrokomputera. W wypadku gdy wymagane jest większe oddalenie lub gdy terminale muszą pracować w warunkach dużych zakłóceń radioelektrycznych możliwe jest zastosowanie łącza z pętlą prądową, umożliwiające oddalenie terminala do około 600 m. Szybkość transmisji w każdym z sześciu kanałów jest programowana niezależnie i wynosi maksymalnie 9600 bitów/s. Złącze drukarek przeznaczone jest do połączenia drukarek według systemu Centronix.

MODUŁ UKŁADU KONTROLI ZASILANIA M-UKZ

Moduł M-UKZ układu kontroli zasilania jest układem bezpośrednio współpracującym z zasilaczami kasety Elwro 800.

Moduł ten realizuje następujące funkcje związane z pracą zasilaczy:

- włączanie i wyłączanie zasilania rozprowadzanego przez magistralę,
- generowanie sygnałów sterujących magistralą systemową, związanych z zanikiem zasilania,
- generowanie sygnałów zerowania INIT/ po załączeniu zasilaczy lub w wyniku naciśnięcia odpowiedniego przycisku pulpitu technicznego,
- generowanie sygnału RESET/ do indywidualnego zerowania wybranych modułów,
- sygnalizację startu systemu, która umożliwia rozróżnienie, czy jest to start częściowy po zaniku zasilania (ang. *warm start*), czy całkowity wskutek wyłączenia systemu (ang. *cold start*).

Ponadto moduł M-UKZ zawiera terminatory rezystorowe poszczególnych linii magistrali oraz układy arbitrażu magistrali systemowej i układ kontroli przekroczenia czasu cyklu adresowania magistrali systemowej.

Układy systemu arbitrażu magistrali systemowej zawierają generator kwarcowy dla sygnałów zegarowych magistrali (CCLK i BCLK) oraz koder i dekoder dla arbitrażu równoległego magistrali.

Układ kontroli przekroczenia czasu cyklu adresowania zabezpiecza system przed zawieszeniem w wyniku zaadresowania nieistniejącej pamięci lub urządzenia wejścia-wyjścia. Zgodnie ze standardem IEC-AMS każdy cykl adresowania na magistrali systemowej może zakończyć się dopiero po uzyskaniu potwierdzenia przesłania danych (XACK) z modułu zaadresowanego. Jeżeli moduł mikrokomputera zaadresuje pamięć lub urządzenie wejścia-wyjścia, którego nie ma w systemie, to wskutek braku potwierdzenia XACK, praca danego modułu mikrokomputera i magistrali systemowej zostaje wstrzymana. Omawiany układ kontroli wykrywa takie wypadki i generuje odpowiedni sygnał (BTMO) umożliwiając przerwanie zawieszonego cyklu adresowania.

SOFTARG '88

Zdumiewająco szybki rozwój rynku sprzętu mikrokomputerowego w Polsce doprowadził w bieżącym roku do prawdziwego zalewu imprez informatycznych o charakterze handlowym. Musiało to oczywiście wywołać wyraźny ich przesyt, odczuwany już nie tylko w środowisku profesjonalnym. Sądzę jednak, że oceny takiej nie można odnieść do III Ogólnopolskich Targów Oprogramowania SOFTARG '88, jakie odbyły się w dniach 20-23 września w Katowicach. Warto bowiem podkreślić szczególnie pożyteczną w warunkach krajowych specyfikę tej imprezy oraz fakt, że mimo gigantycznych rozmiarów aktualnej oferty światowego przemysłu softwarowego, właściwie nie ma ona znaczących zagranicznych odpowiedników.

Pomysł Ogólnopolskich Targów Oprogramowania, jako organizowanej co dwa lata cyklicznej imprezy, powstał w Katowicach i został zrealizowany po raz pierwszy w 1979 r. Załamanie się polskiej informatyki na początku lat osiemdziesiątych spowodowało siedmioletnią przerwę w organizowaniu Targów. Reanimacja imprezy nastąpiła dopiero w wyniku rewolucji mikrokomputerowej, która nie tylko spowodowała przemieszczenie punktu ciężkości zainteresowań rynku, ale również doprowadziła w 1986 r. do zorganizowania przez Ośrodek Postępu Technicznego w Katowicach II Ogólnopolskich Targów Oprogramowania. Ta sama instytucja była również organizatorem tegorocznych III Targów.

Zgodnie z ogłoszonym programem Targi oprócz typowych funkcji handlowych mają na celu spełnienie istotnych funkcji zapotrzebowania społecznego, jakim jest prezentacja możliwie najpełniejszej krajowej oferty systemowych, narzędziowych i użytkowych produktów programowych, promocja wyróżniających się produktów programowych oraz skatalogowanie produktów powielarnych. Prezentacja jest realizowana w różnych formach: zgłoszenia opisu produktu do katalogu, pokazu na sprzęcie własnym lub udostępnionym przez organizatorów oraz wygłoszenia referatu na towarzyszących imprezie seminariach tematycznych. Warunkiem zgłoszenia produktu do prezentacji była jego dostępność w sensie handlowym w ciągu najbliższego roku.

W kolejnych dniach SOFTARG '88 odbyły się następujące seminaria: sterowanie i pomiary, ochrona oprogramowania, systemy CAD/CAM oraz zastosowania do celów zarządzania. Wydany w trzech tomach formatu A5 katalog ujął 1074 produkty programowe, zgłoszone przez 174 jednostki autorskie. Chociaż jest on niewątpliwie największym krajowym przeglądem oprogramowania, to wspomniana liczba jednostek autorskich świadczy o tym, że opublikowane produkty stanowią jedynie cząstkę krajowego dorobku w tej dziedzinie. Ujawniony fakt komputerowego zarejestrowania treści katalogu stwarza nie tylko możliwość szybkiej jego aktualizacji, ale w pierwszym rzędzie jego rozszerzenia o wiele wartościowych, a dotąd nie ujawnionych produktów. O ile ekspozycja ograniczała się wyłącznie do prezentacji na sprzęcie mikrokomputerowym (przy dominacji sprzętu kompatybilnego z IBM PC), to w katalogu jeszcze ok. 15 % pozycji dotyczyło oprogramowania komputerów dużych i średniej wielkości (IBM, Riad, ICL, Odra) oraz ok. 10% – minikomputerów (SM, PDP).

Zgodnie z tradycją poprzednich imprez, podczas SOFTARG'88 zorganizowano również konkurs na najlepsze produkty programowe. 56 wystawców, co stanowiło ok. połowę łącznej ich liczby, zgłosiło 126 produktów programowych w następujących kategoriach oprogramowania: systemowego i narzędziowego, do celów zarządzania oraz do celów specjalnych. Przedmiotem oceny jury konkursu były: demonstracja działania produktu na komputerze, poziom dokumentacji eksploatacyjnej oraz lista zrealizowanych zastosowań.

W pierwszej kategorii produktów nie przyznano żadnej nagrody pierwszego stopnia, natomiast drugie nagrody uzyskały: generator aplikacji SUPER firmy MacroSoft oraz nakładka do systemu zarządzania plikami DAGGER, opracowana przez Meraster. Trzecią nagrodę przyznano wielostanowiskowej nakładce na system operacyjny ADOS firmy MicroSystem. Wyróżnienia w tej kategorii uzyskały: system sieciowy Mikrolan firmy CSK oraz pakiet wspomagania prac projektowych MIX PAK Politechniki Śląskiej.

W drugiej kategorii również nie przyznano pierwszej nagrody. Drugą nagrodę uzyskała firma INTERAMS za rozbudowany system programów do zarządzania przedsiębiorstwem, a trzecią – firma Procom za program F-K. Wyróżnienia przyznano: firmie MicroSystem za kompleksowy system programów zarządzania przedsiębiorstwem, spółdzielni ALPHA za system rejestracji czasu pracy oraz firmie CompuSoft za system płacowy.

W trzeciej kategorii przyznano dwie pierwsze nagrody: firmie Mikrograf za system wydawniczy PL-Druk oraz firmie O.K. za pakiet O.K. MES/2. Drugą nagrodę otrzymała Spółdzielnia PROSTER za pakiet NEUROSCAN, a dwie trzecie – ZETO Wrocław za program „historia położnicza" oraz COIG za programy „wentylacja kopalni" oraz „analiza wybuchowości". Przyznano również 7 wyróżnień.

W.K.

MACIEJ M. SYSŁO
Instytut Informatyki
Uniwersytet Wrocławski

POLSKIE TOWARZYSTWO INFORMATYCZNE

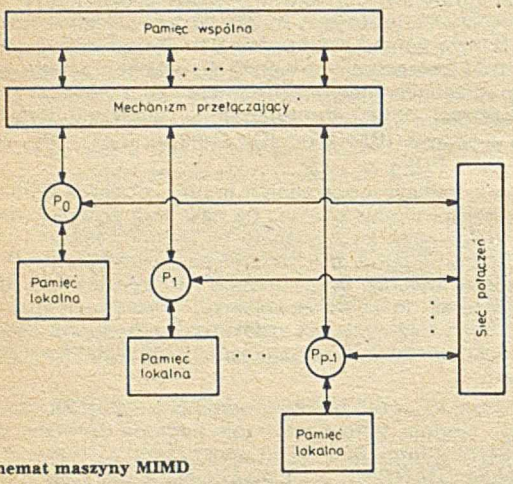
Maszyny i algorytmy równoległe (2)

Maszyny MIMD i SIMD

MASZYNY MIMD

Maszyna MIMD, zwana także asynchronicznym wieloprocesorem (ang. *multiprocessor*), składa się z wielu niezależnych procesorów, które mają dostęp do wspólnej pamięci, oraz wspólnych urządzeń wejścia-wyjścia przez centralny mechanizm przełączający (ang. *switching network*). Każdy z procesorów może dysponować lokalną pamięcią, do której nie ma dostępu żaden inny procesor. Działanie maszyny MIMD jest sterowane przez jeden system operacyjny, umożliwiający komunikację między procesorami na różnych poziomach. Schemat maszyny MIMD przedstawiono na rysunku 1. Wysokie koszty budowy sieci przełączającej ograniczają liczbę procesorów składających się na maszynę MIMD. Maszyny, o których piszę dalej, mają odpowiednio – 16 i 8 procesorów.

Różny stopień zwartości sieci połączeń między procesorami pozwala włączać do rodziny maszyn MIMD także systemy rozproszone, a nawet sieci komputerowe. Obiekty te wykluczono jednak z rozważań, gdyż składają się one zwykle z wielu samodzielnych komputerów, natomiast w przyjętym w artykule rozumieniu komputer równoległy jest pojedynczą maszyną wyposażoną w wiele procesorów.



Rys. 1. Schemat maszyny MIMD

Pierwszą maszyną typu MIMD była D-825 zbudowana przez Burroughs Corp. w 1962 r. Najbardziej znanymi maszynami MIMD są C.mmp i HEP. Maszyna C.mmp została zbudowana i zainstalowana w Uniwersytecie Carnegie-Mellon w 1974 r. Składa się z 16 minikomputerów DEC PDP-11 połączonych ze wspólną pamięcią za pomocą krzyżownicy (ang. *crossbar switch network*). Największym odstępstwem od przyjętego wzoru maszyny MIMD jest w C.mmp dołączenie urządzeń wejścia-wyjścia do poszczególnych procesorów w celu ich wyłącznego użytku. Maszyna HEP (ang. *heterogeneous element processor*) zbudowana przez Denelcor pod koniec lat siedemdziesiątych składa się z ośmiu identycznych procesorów.

Algorytm na maszynę typu MIMD (zwany w skrócie algorytmem MIMD) można przedstawić jako zbiór współpracujących ze sobą procesów, które mogą być wykonywane jednocześnie. Należy odróżnić pojęcie procesu od pojęcia procesora. Proces oznacza wykonanie pewnych obliczeń przez jeden procesor, procesor zaś jest jednostką sprzętową zdolną wykonywać procesy. Przyporządkowaniem procesów procesorom zajmuje się system operacyjny. Podstawowa trudność w opracowaniu algorytmu MIMD leży w tym, iż nie można przewidzieć czasu wykonania poszczególnych kroków procesu, na który może mieć wpływ wiele czynników, takich jak zawartość pamięci, reguły szeregowania stosowane przez system operacyjny i różnice w szybkościach procesorów. Aby zapewnić zgodność wyników obliczeń ze specyfikacją algorytmu, należy zadbać o odpowiednie przeplatanie się procesów. Jedną z metod gwarantujących poprawność obliczeń jest synchronizacja, która może jednak zabierać dużo czasu i ograniczać współbieżność procesów.

Inną zasadniczą kwestią związaną z maszynami MIMD jest rozwiązanie konfliktów między procesorami w czasie jednoczesnego czytania z i (lub) zapisywania do tej samej komórki pamięci. Na ogół zakłada się, że jednocześnie zawartość tej samej komórki może pobierać dowolna liczba procesorów, natomiast w danej chwili co najwyżej jeden procesor może zapisywać w tej samej komórce. To ostatnie ograniczenie można osłabić na wiele różnych sposobów (patrz [10]). Można na przykład dopuścić, by więcej niż jeden procesor jednocześnie zapisywał w tej samej komórce pod warunkiem, że wszystkie procesory umieszczają taką samą wartość. Inna reguła polega na przyjęciu, że efektem jednoczesnego zapisu przez wiele procesorów w tej samej komórce jest to, co zapisywał procesor o najmniejszym numerze.

Współbieżna realizacja algorytmu Warshalla-Floyda

W przedstawionym poniżej opisie realizacji algorytmu Warshalla-Floyda dla maszyny HEP [4] zakłada się, że więcej niż jeden procesor może pobierać równocześnie zawartość tej samej komórki, natomiast dwa procesy nie mogą ani zapisywać, ani odczytywać i zapisywać w tej samej komórce jednocześnie.

Algorytm Warshalla-Floyda wyznacza długości najkrótszych dróg między każdą parą wierzchołków w sieci. Niech $G_d = (V,A;d)$ będzie siecią, w której $d: A \rightarrow R$ jest funkcją długości łuków. O sieci G_d zakłada się jedynie, że nie zawiera cykli ujemnej długości, gdyż w przeciwnym wypadku problem nie miałby skończonego rozwiązania. Dla uproszczenia rozważań można przyjąć, że $A = V \times V$, a jeśli para (i,j) nie jest łukiem w G, to przyjmuje się $d_{ij} = +\infty$. Niech $D = (d_{ij})$ będzie macierzą odległości między wierzchołkami $V = \{1, 2,..., n\}$. Algorytm Warshalla-Floyda (WF) wyznacza ciąg macierzy $D^0 = D, D^1, D^2 ... D^n$ takich, że macierz D^* długości najkrótszych dróg spełnia $D^* = D^n$.

Algorytm WF ma następującą postać:

$$d_{ij}^k = \min \{d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\}$$
$$i, j = 1, 2,..., n,$$

dla $k = 1, 2,..., n$. Łatwo zauważyć, że wykonując obliczenia wierszami, kolejne macierze D^k mogą być tworzone w tych samych komórkach pamięci. Algorytm WF można więc zapisać w postaci:

$$d_{ij} = min\{(d_{ij}, d_{ik} + d_{kj}\}, \quad i, j = 1, 2,..., n,$$

dla $k = 1, 2,..., n$. Co więcej, dla ustalonego k, elementy k-tej macierzy mogą być obliczane niezależnie jeden od dru-

giego. Otrzymujemy zatem następujący równoległy algorytm Warshall-Floyda:

Algorytm PWF

for $k:=1$ **to** n **do**
  for $1 \leqslant i,j \leqslant n$ **do simultaneously**
T_{kij}: **if** $d_{ij} > d_{ik}+d_{kj}$ **then** $d_{ij}:=d_{ik}+d_{kj}$

W celu udowodnienia poprawności algorytmu PWF, można zastosować teorię rozwiniętą dla projektowania i kontroli współbieżnych procesów w systemach operacyjnych (por. rozdz. 2 w [2]); najpierw należy jednak wprowadzić niezbędne pojęcia. Układ zadań $C = (\mathbf{T}, \leqslant)$ składa się ze zbioru zadań (obliczeniowych) i relacji (częściowego porządku) $\leqslant$, przy czym $T<T'$ oznacza, że wykonywanie zadania T' można rozpocząć dopiero po wykonaniu zadania T. Z każdym zadaniem $T \in \mathbf{T}$ są związane dwa podzbiory pamięci: *dziedzina* D_T oraz *zakres* R_T. Zadanie T pobiera wartości z komórek w D_T, a zapisuje w komórkach z R_T. Dwa zadania T i T' są *niekolidujące*, jeśli albo $T<T'$, albo $T'<T$, albo

$$D_T \cap R_{T'} = R_T \cap D_{T'} = R_T \cap R_{T'} = \emptyset$$

Zadania w T są *nawzajem niekolidujące*, jeśli każda para zadań w $\mathbf{T}$ jest niekolidująca. Dowodzi się, że układ złożony z nawzajem niekolidujących zadań jest *zdeterminowany*, tzn. dla ustalonego początku stanu pamięci, jakakolwiek kolejność wykonania zadań kończy się identycznym stanem pamięci. Wracając do algorytmu PWF, jeśli określi się podzbiory nawzajem niekolidujących zadań, to na podstawie podanego wyżej faktu, algorytmy WF i PWF będą wyznaczały identyczne macierze D. Zadanie T_{kij}, dla $k,i,j = 1, 2,..., n$, jest zdefiniowane w ostatniej instrukcji algorytmu PWF, a $C_k = (\mathbf{T}_k, \emptyset)$ jest układem zadań, w którym $\mathbf{T}_k = \{T_{kij}: i,j = 1, 2,..., n\}$ i relacja $\leqslant$ jest pusta. Zatem

$$D_{kij} = \{M_{ij}, M_{ik}, M_{kj}\},\ R_{kij} = \{M_{ij}\}$$

oraz jeśli G_d nie ma ujemnych cykli, to $R_{kkj} = R_{kik} = \emptyset$, gdzie M_{ij} jest komórką pamięci przechowującą d_{ij}. Łatwy dowód bezkolizyjności zadań w układzie C_k pozostawiamy Czytelnikowi.

Algorytm PWF może być wykonywany przez maszynę MIMD, która dysponuje n^2 procesorami. Niestety, istniejące maszyny MIMD dysponują bardzo ograniczoną liczbą procesorów. Przedstawiona realizacja algorytmu PWF będzie dotyczyć maszyny z p procesorami (konkretnie maszyny HEP, dla której $p = 8$). Zakłada się zatem, że maszyna może wykonywać co najwyżej p procesów jednocześnie, a komunikacja między procesorami odbywa się za pomocą wspólnej pamięci. W Fortranie dla maszyny HEP są dostępne następujące instrukcje umożliwiające programowanie algorytmów równoległych: **create, lock** i **unlock.** Jeśli proces P_1 wykonuje instrukcję

create P_2

to proces P_2 rozpoczyna swoje działanie i P_1 jest nadal kontynuowany. Dla każdej komórki pamięci X, po wykonaniu przez proces P_1 instrukcji

lock X

każdy inny proces, który usiłuje odczytać, zapisać lub wykonać **lock** X musi poczekać, aż P_1 wykona instrukcję

unlock X

Algorytm PWF-HEP wykonuje $p-1$ razy instrukcję **create** w celu utworzenia p współbieżnych procesów (proces MASTER i $p-1$ procesów WORKER) i używa poleceń **lock** i **unlock** do zapewnienia synchronizacji między tymi procesami. Proces WORKER(„l") wyznacza wiersze l-ty, $(l+p)$-ty, $(l+2p)$-ty,... kolejnych macierzy D^k. Zmienna *SYN* gwarantuje, że żaden proces nie może przejść do obliczania elementów macierzy D^{k+1}, jeśli któryś z procesów nie zakończył jeszcze wyznaczania elementów macierzy D^k.

Algorytm PWF-HEP
Proces MASTER
  $SYN:=0$
  for $l:=1$ **to** $p-1$ **do create** WORKER(l);
  execute WORKER(p).
Proces WORKER(l)
  for $k:=1$ **to** n **do begin**
    for $i;\ =\ l$ **step** p **until** n **do**
      if $d_{ik} < \infty$ **then**
        for $j:=1$ **to** n **do execute** T_{kij};
    lock *SYN*; $SYN:=SYN+1$; **unlock** *SYN*;
$L1$: **if** $SYN < p \cdot k$ **then go to** $L1$
  end.

Aby poprawnie interpretować działanie algorytmu PWF-HEP należy zauważyć, że instrukcja zwiększenia wartości zmiennej *SYN* o 1 w trzecim od końca wierszu jest wykonywana w danej chwili przez co najwyżej jeden procesor. Instrukcja w następnym wierszu może być natomiast wykonywana przez wszystkie procesy jednocześnie i jej efektem jest równoczesne zakończenie wszystkich procesów.

Eksperymenty obliczeniowe przeprowadzone z algorytmem PWF-HEP na maszynie HEP wykazały jego dużą efektywność zwłaszcza dla pełnych sieci G_d i przypadków, gdy liczba wierzchołków w sieci była wielokrotnością liczby procesorów, gdyż wówczas wszystkie procesory są obciążone prawie równomiernie [4].

Podobne podejście zastosowano w wielu pracach autorów z Washington State University w Pullman do rozwiązywania problemów algebraicznych i równań różniczkowych na maszynie HEP. Warto polecić zwłaszcza pracę [9], gdzie układ zadań związanych z rozwiązywaniem układu równań liniowych, bardziej rozbudowany niż powyżej, dopuszcza optymalizację stopnia współbieżności przez konstrukcję tzw. maksymalnie równoległej relacji poprzedzania między zadaniami.

Sortowanie na maszynach MIMD

Algorytmy sortujące dla maszyn MIMD w większości wypadków najpierw wyznaczają rząd (ang. *rank*) lub po prostu numer każdego elementu, a następnie przesuwają elementy w odpowiednie miejsca. Obliczanie numeru danego elementu odbywa się przez wyliczenie (ang. *enumeration*) liczby wszystkich elementów mniejszych od niego. Ze względu na wymaganą dość dużą liczbę procesorów są to algorytmy mało praktyczne. Zilustrowano to na przykładzie algorytmu podanego przez Hirschberga [7] wymagającego $O(n^{1,5})$ procesorów o swobodnym dostępie do wspólnej pamięci i mającego złożoność $O(\log n)$. Odstępstwem od zdefiniowanego w artykule modelu maszyny MIMD jest założenie o synchronicznym działaniu procesorów (por. kroki 3 i 6 w opisie algorytmu).

W krokach 2 i 5 jest wykonywane sumowanie elementów ciągów $\sqrt{n}$-elementowych. Ogólnie, suma n elementów a_1 może być znaleziona na synchronicznej maszynie MIMD z n procesorami za pomocą następującego algorytmu:

Algorytm PSUMA

for $i:\ =\ 0$ **to** $\lceil \log n \rceil - 1$ **do**
  for $1 \leqslant j \leqslant n - 2^i$ **do simultaneously**
    $a_j:=a_j\ +\ a_{j+2^{**}i}$
{a_1 jest sumą n elementów a_i}.

Złożoność algorytmu PSUMA wynosi $\lceil \log n \rceil$. Inna metoda sumowania n liczb na maszynie MIMD wymaga jedynie $n/\log n$ procesorów, a jej czas działania wynosi $2\lceil \log n \rceil$. Metoda ta składa się z dwóch kroków. W pierwszym kroku, każdy z procesorów znajduje sumę przydzielonego mu podciągu $(\log n)$-elementowego, a w następnym kroku, znalezione sumy częściowe są kumulowane metodą binarną. Dowód poprawności tej metody oraz podanej złożoności pozostawiono Czytelnikowi (por. [6]). W powyższych rozważaniach dodawanie może być zastąpione przez każde inne działanie łączne, takie jak mnożenie, branie minimum lub maksimum, suma i iloczyn boolowski itp.

Algorytm Hirschberga

Zakłada się, że wszystkie elementy ciągu $a_1, a_2,..., a_n$ są różne, a n jest kwadratem liczby naturalnej.

Krok 1. Podzielić zbiór n liczb na $\sqrt{n}$ grup złożonych z $\sqrt{n}$ liczb każda.
Następujące operacje wykonywać równolegle w każdej grupie elementów.

Krok 2. Dla każdego j, określić rząd COUNT1$_j$ elementu a_j w grupie, do której należy, tj. liczbę tych i, dla których $a_i < a_j$ plus 1. W tym celu, każdemu j przydziela się $\sqrt{n}$ procesorów, zatem $n \times n^{1/2}$ procesorów jest potrzebnych dla wszystkich elementów. Wszystkie porównania między elementami a_i mogą być współbieżnie wykonane w jednostce czasu, a wyniki porównań mogą być skumulowane w czasie $\lceil \log \sqrt{n} \rceil$.

Krok 3. Każdą grupę uporządkować zgodnie z wartościami COUNT1, tj. jednocześnie wykonać a COUNT1j:= a_j. Porządek ten może być znaleziony w stałym (tj. niezależnym od n) czasie i nie pojawi się żaden konflikt dostępu do pamięci, gdyż wartości COUNT1 są różne (w grupach) dla różnych elementów.

Krok 4. Przydzielić każdemu elementowi a_j zbiór $\sqrt{n}$ procesorów. Procesory te równocześnie wykonują binarne przeszukiwanie uporządkowanych $\sqrt{n}$ grup i określają następujące wartości

$$\text{COUNT2}_{j,y} = \begin{cases} j, \text{ jeśli } a_j \text{ należy do grupy } y, \\ \text{liczba elementów } a_i \text{ w grupie } y \text{ takich, że } a_i < a_j. \end{cases}$$

Wartości COUNT2$_{j,y}$ mogą być znalezione dla wszystkich elementów a_j w czasie $\lceil \log \sqrt{n} \rceil$ przez $n \times n^{1/2}$ procesorów.

Krok 5. Używając $\sqrt{n}$ procesorów dla każdego elementu j, wyznaczyć rząd COUNT$_j$ elementu a_j w całym ciągu, gdzie COUNT$_j = \Sigma_y$ COUNT2$_{jy}$. Krok ten może być współbieżnie zrealizowany w czasie $\lceil \log \sqrt{n} \rceil$.

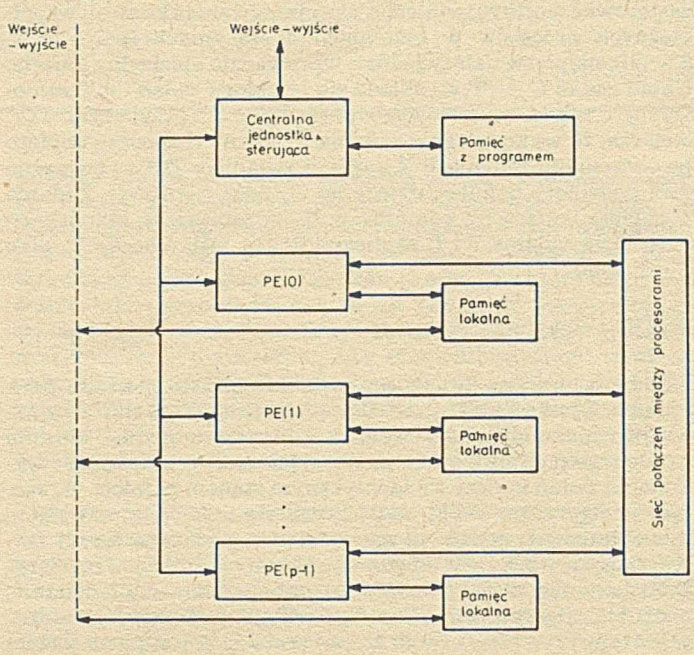
Krok 6. Na podstawie wartości COUNT, utworzyć poszukiwane uporządkowanie. Może to być wykonane współbieżnie w stałym czasie.

Działanie algorytmu Hirschberga dla ciągu (3,7,1,4,6,8,9,5,2) jest następujące. W kroku 1, cały ciąg jest dzielony na trzy grupy (3,7,1|4,6,8|9,5,2). W następnym kroku każdemu elementowi a_j są przydzielone trzy procesory, które najpierw porównują a_j z pozostałymi elementami grup, a następnie sumują wyniki tych porównań, by obliczyć rząd elementu w grupie. Otrzymuje się więc najpierw 27 wyników porównań (001,101,000|000,100,110|011,001,010), a następnie wartości COUNT1 = (2,3,1|1,2,3|3,2,1), które określają miejsca elementów w porządkowanych grupach. W kroku 3 otrzymuje się zatem ciąg (1,3,7|4,6,8|2,5,9). W kroku 4 ponownie korzysta się ze wszystkich 27 procesorów, by metodą binarnego przeszukiwania policzyć wartości COUNT2 = (100,201,322|211,222,332|101,212,333). W następnym kroku sumuje się wartości COUNT2 przyporządkowane poszczególnym elementom, co daje w rezultacie rzędy COUNT = (1,3,7|4,6,8|2,5,9), wyznaczające miejsca elementów z uporządkowanych grup w ciągu całkowicie uporządkowanym (1,2,3,4,5,6,7,8,9).

Algorytm Hirschberga jest najszybszym algorytmem MIMD, rozrzutnym jednak jeśli chodzi o liczbę procesorów. Preparata podał algorytm równie szybki, ale wymagający tylko n log$n$ procesorów. Valiant, a później Gavril podali algorytmy MIMD używające $n/2$ procesorów o czasach działania $0(\log n \log\log n)$ i $0(\log^2 n)$, odpowiednio. Szczegółowe opisy tych algorytmów można znaleźć w pracach przeglądowych [1] i [8].

MASZYNY SIMD

Maszyna SIMD, zwana także macierzą procesorów (ang. *array processors*), składa się ze zbioru procesorów PE, które są zsynchronizowane strumieniem rozkazów (operacji) wysyłanym i sterowanym przez jednostkę centralną (rys. 2). Procesory wykonują rozkazy na danych przechowywanych we własnej (lokalnej) pamięci operacyjnej. Zatem maszyna SIMD może wykonywać ciąg tych samych rozkazów jednocześnie na wielu różnych danych. O każdym z PE zakłada się, że zna i może rozpoznać swój numer. Ponadto, można stosować maski, umożliwiające uaktywnianie tylko pewnych procesorów. Najczęściej zbiór procesorów aktywnych jest określany za pomocą prostego kryterium wyboru, którym może być warunek logiczny umieszczany w algorytmach po rozkazie. Dany rozkaz jest wykonywany przez te procesy, których numery spełniają kryterium. Dla przykładu, jeśli algorytm ma zwiększać zawartość rejestru $A(i)$ w procesorach o parzystym numerze i, to odpowiedni warunek może mieć postać $(i_0 = 0)$, gdzie i_0 jest najmniej znaczącym bitem w dwójkowym rozwinięciu i.



Rys. 2. Schemat maszyny SIMD

Procesory w maszynie SIMD mogą komunikować się między sobą za pomocą albo wspólnej pamięci (SIMD-SM, ang. *shared memory*), albo sieci połączeń (ang. *interconnection network*), która z kolei może przyjmować wiele różnych postaci, takich jak krata (MC, ang. *mesh-connected network*), kostka (CC, ang. *cubeconnected network)* lub sieć przetasowana (PF, ang. *perfect shuffle network*).

Przykładem maszyny SIMD jest Illiac IV, zbudowany w 1972 r. przez korporację Burroughs, złożony z 64 PE tworzących kratę.

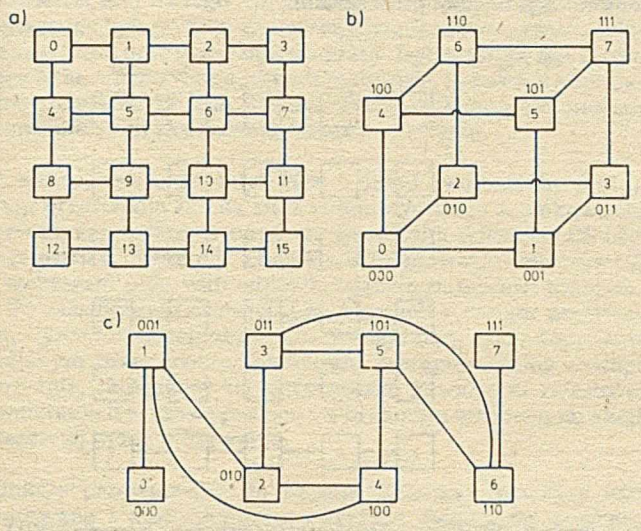
W maszynie SIMD-SM procesory komunikują się przez wspólną pamięć. Zatem przesłanie liczby r z procesora PE(i) do PE(j) następuje w dwóch krokach: PE(i) zapisuje r w komórce c należącej do wspólnej pamięci, a PE(j) odczytuje r z c. Czas komunikacji jest więc stały. Zakłada się zwykle, że co najwyżej jeden procesor może jednocześnie zapisywać w tej samej komórce, natomiast odczytanie zawartości komórki może, ale nie musi, być obwarowane takim samym ograniczeniem. Istnieje dość obszerna literatura poświęcona algorytmom dla maszyn SIMD-SM. Ten model obliczeń jest jednak mało realistyczny ze względu na założenie o możliwości jednoczesnego dostępu przez p procesorów do p komórek pamięci. W praktyce, pamięci są dzielone na bloki, do których może mieć dostęp jednocześnie co najwyżej jeden procesor, zatem realizacja q odwołań do tego samego bloku pamięci trwa $0(q)$ jednostek czasu. Zauważmy, że maszynę SIMD-SM można uważać za szczególną wersję zsynchronizowanej maszyny MIMD.

W maszynie SIMD-MC, procesory są ułożone w k-wymiarową tablicę $P(n_0, n_1, \ldots, n_{k-1})$, gdzie n_i jest rozmiarem i-tego wymiaru oraz $p = \prod_{i=0} n_i$. Procesor PE w pozycji P $(i_0,\ldots,i_{k-1})$ jest połączony z procesorami w pozycjach P $(i_0,\ldots, i_j \pm 1,\ldots, i_{k-1})$ dla $0 \leqslant j \leqslant k$, jeśli tylko istnieją. Zatem prawie każdy procesor jest połączony z 2k innymi procesorami. Na rysunku 3a przedstawiono SIMD-MD z 16 procesorami dla $k = 2$.

Niech $p = 2^q$ i niech $i = [i_{q-1},\ldots, i_0]$ będzie rozwinięciem dwójkowym liczby i, $i \in [0, p-1]$. Niech $\bar{i}$ oznacza dopełnienie bitu i_j, a $i^{(b)} = [i_{q-1},\ldots, i_{b+1}, \bar{i}_b, i_{b-1},\ldots, i_0]$. W maszynie SIMD-

-CC zakłada się, że każdy procesor i jest połączony z procesorami PE ($i^{(b)}$), przy czym $0 \leqslant b < q$. Zatem każdy PE (i) jest połączony z $\log p = q$ innymi procesorami. Na rysunku 3b przedstawiono SIMD-CC z 8 procesorami (tj. $q = 3$).

Niech p, q, i, $i^{(b)}$ będą zdefiniowane jak wyżej. Niech $S(i)$ (ang. *shuffle*) oraz $US(i)$ (ang. *unshuffle*) będą zdefiniowane jako liczby naturalne o rozwinięciach odpowiednio $[i_{q-2}, i_{q-3},..., i_0, i_{q-1}]$ oraz $[i_0, i_{q-1}, i_{q-2},..., i_2, i_1]$. W maszynie SIMD-PS, procesor PE(i) jest połączony z procesorami o numerach $i^{(0)}$, $S(i)$ oraz $US(i)$. Każdy procesor w tym modelu jest połączony z trzema procesorami. Na rysunku 3c przedstawiono komputer SIMD-PS z 8 procesorami.



Rys. 3. Przykłady sieci połączeń w maszynach SIMD

Mnożenie macierzy

Działania na macierzach, ze względu na jawnie tkwiącą w nich równoległość operacji, stanowią najobszerniejszą i najpopularniejszą grupę zadań realizowanych na maszynach SIMD. W tej części artykułu skupiono się na wyznaczaniu iloczynu dwóch dowolnych macierzy. Działanie to ma bardzo dużo zastosowań. Wiele problemów teorii grafów, dla których metody macierzowe nie dostarczają najefektywniejszych algorytmów sekwencyjnych, ma bardzo efektywne algorytmy równoległe, wykorzystujące iloczyn dwóch macierzy (lub pewną jego modyfikację). Do problemów takich należą: wyznaczanie długości najkrótszych dróg sieci, wyznaczanie przechodniego domknięcia, obliczanie promienia, średnicy i centrum grafu, generowanie drzewa częściowego metodą rozszerzania i topologiczne sortowanie acyklicznego digrafu (szczegóły można znaleźć w [3] a także w [5]).

Iloczyn $C = A \times B$ dwóch macierzy stopnia n może być obliczony w czasie $O(n)$ na wyidealizowanym komputerze SIMD-SM z n^2 procesorami. Znane są także algorytmy równoległe o czasie działania $O(\log n)$. Algorytm będący realizacją klasycznej metody mnożenia potrzebuje $n^3/\log n$ procesorów, a współbieżna implementacja algorytmu Strassena – wymaga $n^{\log 7/\log n}$ procesorów. Wszystkie trzy algorytmy osiągają optymalne (z dokładnością do stałego współczynnika) przyśpieszenie.

Jako przykład przedstawiono algorytm Cannona mnożenia dwóch macierzy $A = (a_{ij})$ i $B = (b_{ij})$ stopnia n na maszynie SIMD-MC z $k = 2$, o której dla uproszczenia zakłada się, że jest wyposażona dodatkowo w połączenia odpowiednich procesorów brzegowych, tak aby elementy w wierszach i kolumnach siatki procesorów mogły być przesuwane cyklicznie. Zakłada się także, że każdy procesor PE (i, j) ma trzy rejestry $A(i, j)$, $B(i,j)$ oraz $C(i, j)$. Dane macierze A i B są umieszczone odpowiednio w rejestrach A i B, a wynik ma być pozostawiony w rejestrach C. Algorytm składa się z dwóch faz. W pierwszej fazie, elementy i-tego wiersza macierzy A są przesuwane i razy cyklicznie w lewo, a elementy j-tej kolumny macierzy B – j razy cyklicznie w górę. W rezultacie otrzymuje się $A(i, j) = a_{i,(j+i) \bmod n}$ oraz $B(i, j) = b_{(i+j) \bmod n, j}$, a zatem $A(i, j) * B(i,j)$ jest składnikiem sumy tworzącej c_{ij}. W drugiej fazie każde przesunięcie elementów w rejestrach A i B „nakłada" odpowiednie elementy macierzy A i B na siebie, dając kolejny składnik elementów iloczynu. W poniższym opisie algorytmu symbol „:=" oznacza podstawienie wykonywane w jednym procesorze, a „←" jest poleceniem przesłania informacji po bezpośrednim połączeniu sieci komunikacyjnej.

```
-----------------------------------------------------
begin
    for i:=1 to n-1 do begin
        A(i,j) <- A(i, (j+i) mod n);      (i>=1)
        B(i,j) <- B((i+1) mod n, j)       (j>=1)
    end;
    C(i,j) := A(i,j)*B(i,j);
    for i:=1 to n-1 do begin
        A(i,j) <- A(i, (j+1) mod n);
        B(i,j) <- B((i+1) mod n, j);
        C(i,j) := C(i,j) + A(i,j)*B(i,j)
    end
end
-----------------------------------------------------
```

Wydruk 1. Algorytm Cannona

```
-----------------------------------------------------
                    Faza 1
a00 a01 a02                 b00 b11 b22
a11 a12 a10                 b10 b21 b02
a22 a20 a21                 b20 b01 b12
-----------------------------------------------------
                    Faza 2
a01 a02 a00                 b10 b21 b02
a12 a10 a11                 b20 b01 b12
a20 a21 a22                 b00 b11 b22
-----------------------------------------------------
                    Faza 3
a02 a00 a01                 b20 b01 b12
a10 a11 a12                 b00 b11 b22
a21 a22 a20                 b10 b21 b02
-----------------------------------------------------
```

Wydruk 2. Nakładanie się macierzy w kolejnych krokach algorytmu Cannona

Na wydruku 2 zilustrowano algorytm Cannona na przykładzie macierzy stopnia 3.

Całkowity czas komunikacji między procesorami wynosi w algorytmie Cannona $4(n - 1)$ jednostek, a całkowita złożoność algorytmu jest $O(n)$. Kosztem zwiększenia stałej w oszacowaniu złożoności, algorytm Cannona może być zmodyfikowany dla maszyny SIMD-MC bez dodatkowych połączeń procesorów brzegowych (por. [3]). Klasyczny algorytm mnożenia dwóch macierzy ma bardzo prostą implementację w postaci algorytmu systolicznego.

```
-----------------------------------------------------
begin
    for i:=3q-1 downto 2q do begin
        A(m(i)) <- A(m);                  (m_i = 0)
        B(m(i)) <- B(m)                   (m_i = 0)
    end; {A(i,j,k)=a_ik, B(i,j,k)=b_jk dla i,j,k=0,...,n-1}
    for i:=q-1 downto 0 do {A(i,j,k) do A(i,j,*)}
        A(m(i)) <- A(m);                  (m_i = m_2q+i)
    for i:=2q-1 downto q do {B(i,j,k) do B(i,*,k)}
        B(m(i)) <- B(m);                  (m_i = m_q-1)
    C(m) := A(m)*B(m); {C(i,j,k)=a_ji b_ik dla i,j,k=0,...,n-1}
    for i:=2q to 3q-1 do
        C(m) <- C(m) + C(m(i))
end
-----------------------------------------------------
```

Wydruk 3. Algorytm mnożenia macierzy dla maszyny SIMD-CC

Poniżej opisano algorytm mnożenia dwóch macierzy dla maszyny SIMD-CC (wydruk 3) z n^3 procesorami, gdzie $n = 2^q$. Dla uproszczenia oznaczeń zakłada się, że procesory tworzą siatkę $n \times n \times n$, czyli PE(i,j,k) ma numer $in^2 + jn + k$, gdzie $i,j,k \in [0, n-1]$. Zatem, jeśli $r_{3q-1},..., r_0$ jest rozwinięciem dwójkowym numeru procesora PE(i,j,k), to $i = r_{3q-1},..., r_{2q}$, $j = r_{2q-1},..., r_q$ oraz $k = r_{q-1},..., r_0$. Tak jak poprzednio, każdy procesor ma trzy rejestry A, B oraz C; na początku $A(0,j,k) = a_{jk}$, $B(0,j,k) = b_{jk}$, a wynik ma być umieszczony w $C(0,j,k) = c_{jk}$, dla $j,k \in [0, n-1]$. Algorytm składa się z dwóch faz. W pierwszej fazie, elementy macierzy A i B są rozsyłane do procesorów tak, aby $A(l,j,k) = a_{jl}$ oraz $B(l,j,k) = b_{lk}$. W drugiej fazie są kumulowane składniki elementów iloczynu.

Warto prześledzić jedynie działanie ostatniej instrukcji for, która w każdym rejestrze $C(0,j,k)$ kumuluje sumę $c_{jk} = \sum_{i=0}^{n-1} a_{ji} b_{ik}$.

Ponieważ zmienna sterująca l przebiega od $2q$ do $3q-1$, zmiana bitu w $m^{(l)}$ następuje na pozycji pierwszego indeksu. Dla $l=2q$, otrzymuje się $C(i,j,k) \leftarrow C(i,j,k) + C(i+1,j,k)$ dla parzystych i oraz $C(i,j,k) \leftarrow C(i,j,k) + C(i-1,j,k)$ dla nieparzystych i. Dla $l=2q+1$ jest $C(i,j,k) \leftarrow C(i,j,k) + C(i+2,j,k)$, gdy drugi najmniej znaczący bit w rozwinięciu i jest 0, oraz $C(i,j,k) \leftarrow C(i,j,k) + C(i-2,j,k)$, w przeciwnym wypadku. Dla $l = 2q$, rejestr $C(2,j,k)$ otrzymuje wartość $C(2,j,k) + C(3,j,k)$, a zatem po dwóch iteracjach rejestr $C(0,j,k)$ zawiera sumę $C(0,j,k) + C(1,j,k) + C(2,j,k) + C(3,j,k)$, czyli liczba składników w $C(0,j,k)$ jest podwajana w każdej iteracji. W rezultacie, po q iteracjach rejestr $C(0,j,k)$ zawiera sumę

$$\sum_{i=0}^{n-1} C(i,j,k) = c_{jk}.$$

Powyższy algorytm dla maszyny SIMD-CC z n^3 procesorami ma złożoność $O(q)$, a więc $O(\log n)$. W pracy [3] zawarto wiele innych algorytmów mnożenia macierzy dla maszyn SIMD-CC, a wśród nich algorytm dla n^2 procesorów o złożoności $O(n)$ oraz dla n^2m procesorów o złożoności $O(n/m + \log m)$, gdzie m spełnia $1 \leqslant m \leqslant n$. Oba algorytmy są optymalne, w drugim wypadku należy przyjąć $m = n/\log n$. Podobna sytuacja jest także dla maszyn SIMD-PS, odpowiednie algorytmy są jednak bardziej skomplikowane.

Algorytm naprzemiennego sortowania

Do posortowania n liczb, struktura połączeń między procesorami w maszynie SIMD może być liniowa; taką maszynę oznaczać będziemy w skrócie przez SIMD-L ang. *linear*. Działanie maszyny SIMD-L zilustrowano na przykładzie uwspółbieżnienia algorytmu sekwencyjnego, którego czas działania wynosi $O(n^2)$. Rozpatrzmy *metodę bąbelkową* (ang. *bubble sort*), w której porównuje się i w razie potrzeby zamienia miejscami elementy sąsiednie. W pierwszej iteracji algorytmu, element największy jest umieszczany na końcu ciągu, następnie drugi największy itd. Liczba porównań w tej metodzie wynosi co najwyżej

$$\sum_{i=1}^{n-2} (n-i)$$

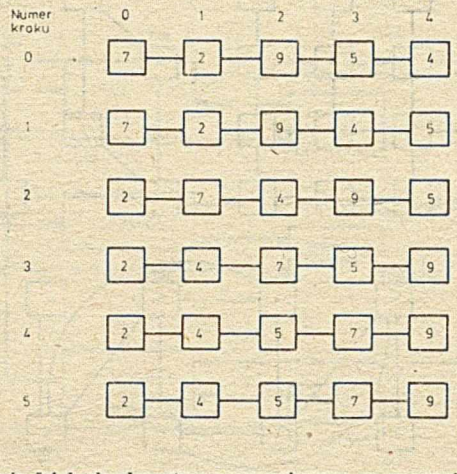
a więc jest rzędu n^2. Odmianą metody bąbelkowej jest metoda przemiennego sortowania (ang. *odd-even transportation sort*) składająca się z n przeplatających się nawzajem faz nieparzystych i parzystych, z których każda wykonuje $n/2$ porównań. W każdej z faz element aktywny x_i jest porównywany ze swoim prawym sąsiadem x_{i+1} i jeśli $x_i > x_{i+1}$, to elementy te są zamieniane miejscami. W fazie nieparzystej, aktywne są elementy o wskaźnikach nieparzystych a w fazie parzystej elementy o wskaźnikach parzystych. Przyjmijmy, że element x_i znajduje się w i-tym procesorze maszyny SIMD-L w rejestrze o tej samej nazwie. Procesor aktywny porównuje zawartość swojego rejestru x z zawartością rejestru x w następnym procesorze i w razie konieczności zamienia miejscami zawartości. Poniższy algorytm naprzemiennego sortowania wymaga n procesorów, a jego czas działania wynosi $O(n)$.

for $l := 1$ **to** n **do**
  if $x_i > x_{i+1}$ **then** EXCHANGE $(x_i, x_{i+1})(i_0 = l_0)$
{i_0 oznacza najmniej znaczący bit w rozwinięciu dwójkowym i, podobnie l_0 w l.}

Na rysunku 4 zilustrowano działanie algorytmu naprzemiennego sortowania na przykładzie ciągu pięcioelementowego. Aktywne połączenia zostały pogrubione.

Podobnie można uwspółbieżnić algorytm sortujący, który wykorzystuje drzewo binarne jako strukturę połączeń między elementami i wynikami porównań. Jako rezultat otrzymuje się inny równoległy algorytm sortujący n liczb w czasie $O(n)$ przy użyciu $O(n)$ procesorów. Szczegółową realizację tego algorytmu pozostawiono Czytelnikowi.

Metoda naprzemiennego sortowania może być uogólniona na wypadek k liniowo połączonych procesorów, z których każdy może przechowywać w swojej pamięci m liczb. Niech $n = km$ i niech każdy z procesorów ma w swojej pamięci m liczb. Na początku każdy procesor sortuje swój podciąg jedną z klasycznych metod sekwencyjnych. W następnych k krokach, uaktywniane są naprzemiennie parzyste i nieparzyste procesory. Procesor aktywny otrzymuje podciąg z procesora następnego, scala oba podciągi w jeden ciąg uporządkowany, dzieli otrzymany ciąg na dwie równe części i drugą z nich odsyła do procesora następnego. Złożoność całego algorytmu jest równa $O(m \log m) + O(km)$, gdzie pierwszy składnik jest oszacowaniem czasu współbieżnego posortowania podciągów w procesorach, a drugi – złożonością kroku iteracyjnego. Oszacowanie złożoności można wyrazić także w postaci $O(n \log n)/k = O(n)$, a zatem dla n dużych w stosunku do k, współczynnik przyspieszenia dla tej metody jest bliski k. To bliskie optymalnego przyspieszenie uzyskuje się przede wszystkim dzięki temu, że dla n dużych w porównaniu z k, czas pracy procesorów dominuje nad czasem komunikacji między procesorami.



Rys. 4. Ilustracja działania algorytmu naprzemiennego na maszynie SIMD-L

Opisy algorytmów sortujących dla maszyn SIMD z innymi sieciami połączeń można znaleźć w [1] i [8]. Jeszcze inne algorytmy sortujące dla maszyn SIMD mogą być otrzymane przez odpowiednią implementację sieci sortujących oraz algorytmów systolicznych, omawianych w następnej części.

LITERATURA

[1] Bitton D. et al.: A taxonomy of parallel sorting. Computing Surveys, Vol. 16, pp. 187–318, 1984

[2] Coffman E. G., Jr., Denning P. J.: Operating Systems Theory. Prentice-Hall, Englewood Cliffs, 1973

[3] Dekel E. et al.: Parallel matrix and graph algorithms. SIAM J. Comput., Vol. 10, pp. 657–675, 1981

[4] Deo N. et al.: Two parallel algorithms for shortest path problems. Technical Report CS, 80–059, Washington State University, Pullman (WA), 1980

[5] Dudziński E.: Wybrane równoległe algorytmy kombinatoryczne. Matematyka stosowana, Tom 28, str. 91–123, 1986

[6] Heller D.: A survey of parallel algorithms in numerical linear algebra. SIAM Review, Vol. 20, pp. 740–777, 1987

[7] Hirschberg D. S.: Past parallel sorting algorithms. Comm ACM, Vol. 21, pp. 657–666, 1978

[8] Laksmivaraham S. et al.: Parallel sorting algorithms. Advances in Computers. Vol. 23, pp. 295–354, 1984, Academic Press

[9] Lord R. E. et al.: Solving linear algebraic equations on an MIMD computer. Journal of the ACM, Vol. 30, pp. 103–117, 1983

[10] Vishkin U.: Synchronous parallel computation – a survey. Niepublikowane, 1983

Uprzejmie informujemy Czytelników, że egzemplarze INFORMATYKI – bieżące i archiwalne – można kupić nie tylko w kioskach Ruchu, Klubie NOT SIGMY, Zakładzie Kolportażu i Dziale Handlowym (szczegóły podano w WARUNKACH PRENUMERATY), ale również

w lokalu naszej redakcji

ul. Mickiewicza 18 m. 17 w Warszawie, tel. 39-14-34

oraz

w specjalistycznej księgarni PP „Domu Książki"

ul. Mokotowska 51/53 w Warszawie, tel. 28-16-14

Zapraszamy wszystkich zainteresowanych.

JAN BIELECKI
Warszawa

Turbo C

Przerwania, sygnały, powroty

Zgłoszenie przerwania jest wyrazem gotowości do wykonania akcji niezależnej od tego, co właśnie wykonuje procesor. Przerwania mogą być generowane przez czynniki zewnętrzne względem programu oraz przez sam program. Jeśli wystąpi zgłoszenie przerwania, a przerwanie jest odblokowane, to nastąpi automatyczne przejście do jego obsługi. Po jej zakończeniu nastąpi wznowienie akcji zawieszonej przez przerwanie.

W mikroprocesorze Intel 8086 i jemu podobnych, prolog obsługi przerwania składa się z automatycznego zablokowania przerwań, zapamiętania na stosie zawartości rejestrów *CS* i *IP* oraz rejestru znaczników, a następnie załadowania rejestrów *CS* i *IP* wektorem przerwań odpowiadającym numerowi przerwania. Po zakończeniu czynności związanych z danym przerwaniem, wyznaczonych przez program znajdujący się pod adresem określonym przez wektor przerwań, następuje epilog obsługi przerwania, polegający na odtworzeniu zawartości rejestrów zapamiętanych w czasie prologu i tym samym wznowieniu akcji zawieszonej przez przerwanie.

Funkcje przewidziane do obsługi przerwań powinny być deklarowane z modyfikatorem *interrupt*. Różnią się wówczas tym od pozostałych funkcji, że w ich prologu zostaną umieszczone na stosie nie tylko zawartości rejestrów *SI, DI* i *BP*, ale również rejestrów *AX, BX, CX, DX, ES* i *DS* (rejestry *CS, IP* i rejestr znaczników zostaną zapamiętane na stosie w prologu obsługi przerwania). W epilogu funkcji nastąpi automatyczne odtworzenie tych dodatkowych rejestrów.

W ogólnym wypadku, deklaracja zapowiadająca funkcje obsługiwania przerwań powinna mieć postać:

```
void interrupt
fun (bp, di, si, ds, es dx, cx, bx, ax, ip, cs, flags,...):
```

Jak wynika z tego zapisu, funkcja może mieć dostęp do wszystkich rejestrów procesora, identyfikowanych w obrębie jej definicji przez nazwy jej parametrów. Ponieważ funkcja może być wywoływana ze zmienną liczbą argumentów, nie jest wymagane ograniczenie się do argumentów przekazanych w rejestrach.

Zainstalowanie wektora przerwań, których obsługę przejmie funkcja zadeklarowana z modyfikatorem *interrupt*, powinno odbyć się dwuetapowo. W pierwszym etapie należy przechować systemowy wektor przerwań, a w drugim zastąpić go nowym wektorem. Realizacja pierwszej z wymienionych czynności odbywa się za pomocą funkcji *getvect*, której argumentem jest numer przerwania, a realizacja drugiej odbywa się za pomocą funkcji *setvect*, której argumentami są: numer przerwania i wyrażenie reprezentujące dalekie wskazanie funkcji obsługującej przerwania o danym numerze.

Przykład

Zainstalowanie wektora przerwań, które powoduje, że cyklicznie generowane przerwania zegarowe o numerze *1C* będą każdorazowo powodować asynchroniczne wykonywanie funkcji *timer*

```
#include <stdio.h>
#include <dos.h>
#define INT 0x1C
static volatile Count, Flag;
void interrupt far
timer ( )
{
 if (Count ++ % 18)
   return;
  Flag ++;
}
main ( )
{
 void interrupt (far * Ref) ( );
 unsigned i = 0;
 Ref = getvect (INT);
 setvect (INT, timer);
 while (!kbhit ( )) {
   printf ("%8u", i ++);
   if (Flag)
      Flag--, putchar ('\a');
 }
 setvect (INT, Ref);
}
```

W systemie MS-DOS, w przybliżeniu 18,2 raza na sekundę jest generowane przerwanie *1C*. Jego zadaniem jest aktualizacja czasu dobowego. Przytoczone w programie zainstalowanie wektora przerwań powoduje, że każdej takiej aktualizacji będzie towarzyszyć wykonanie funkcji *timer*. Spowoduje to, że asynchronicznie do wyprowadzenia kolejnych liczb całkowitych, mniej więcej co 1 s, będzie rozlegał się sygnał dźwiękowy. Wprowadzenie z klawiatury dowolnego znaku spowoduje przerwanie wprowadzania liczb i przywrócenie pierwotnego stanu wektora przerwań.

Jak już wyjaśniono, przerwania mogą być generowane nie tylko przez czynniki zewnętrzne względem programu, ale również przez sam program. W celu uzyskania takiego efektu można posłużyć się funkcją *intr*. Jej deklaracja prototypowa ma postać;

```
void intr (int num, struct REGPACK * regs);
```

i odwołuje się do struktury typu (struct REGPACK) zadeklarowanej w pliku nagłówkowym *dos.h*;

```
struct REGPACK {
                unsigned r_ax, r_bx, r_cx, r_dx;
                unsigned r_bp, r_si, r_di,
                         r_ds, r_es, r_flags;
};
```

Argumentami funkcji *intr* są: numer przerwania i wskazanie struktury typu (struct REGPACK). Wywołanie funkcji *intr* powoduje skopiowanie poszczególnych pól struktury do rejestrów procesora, zgodnie z mnemoniką tych pól, wygenerowanie przerwania o numerze *num*, a następnie odesłanie rejestrów do pól struktury.

Przykład

Wygenerowanie przerwania 0x21 w celu wyprowadzenia napisu Jan Bielecki.

```
#include <stdio.h>
#include <dos.h>
main ( )
{
  char far * Name = "Jan Bielecki $";
  struct REGPACK Str;
```

```
  Str. r_ax = 9 << 8;
  Str. r_ds = FP_SEG (Name);
  Str. r_dx = FP_OFF (Name);
  intr (0 × 21, & Str);
}
```

W rejestrze *AH* umieszczono numer funkcji obsługiwanej przez przerwanie 0×21, a w rejestrach *DS* i *DX* umieszczono odpowiednio: numer segmentu i odstęp w segmencie, określające adres pierwszego znaku wyprowadzanego napisu.

Pewnego rodzaju odmianę przerwań programowych stanowią sygnały. Różnią się one tym od przerwań programowych, że nie uruchamiają mechanizmu obsługi przerwania, lecz powodują jedynie wykonanie akcji, która została określona podczas definiowania sygnału.

Uwaga: Przedstawiona tu implementacja sygnałów w języku Turbo C jest wzorowana na proponowanym standardzie ANSI, ale nie jest z nim zgodna.

Definiowanie sygnału odbywa się za pomocą funkcji *ssignal*, której deklaracja prototypowa ma postać:
int (* ssignal (int sig, int (* action) ( ))) ( );

Argumentem funkcji *ssignal* jest numer definiowanego sygnału oraz wskazanie funkcji określającej rodzaj akcji, która zostanie wykonana w chwili wygenerowania sygnału. Wskazanie może identyfikować dowolną funkcję programu albo może być jednym z dwóch wskazań: *SIG_DFL* i *SIG_IGN*, definiowanych w pliku nagłówkowym *signal.h*. Wskazania te są tak dobrane, że pierwsze z nich jest związane z akcją domniemaną, a drugie jest związane z akcją pustą.

Bezpośrednio po wygenerowaniu sygnału następuje niejawne wykonanie instrukcji:

```
ssignal (sig, SIG_DFL);
```

a następnie wykonanie instrukcji:

```
(* action) (sig);
```

Generowanie sygnału odbywa się za pomocą funkcji *gsignal*, której deklaracja prototypowa ma postać:

```
int gsignal (int sig);
```

Rezultatem funkcji *ssignal* dotyczącej sygnału o numerze *num* jest wskazanie funkcji określającej akcję, która zostałaby wykonana, gdyby zamiast wywołania funkcji *ssignal* posłużono się wywołaniem funkcji *gsignal (num)*. Rezultatem funkcji *gsignal* jest natomiast rezultat funkcji określającej akcję dla sygnału *num*. Jeśli akcja ta jest związana z funkcją * *SIG_DFL*, to rezultatem jest dana o wartości 0, a jeśli jest związana z funkcją * *Sig_IGN*, to rezultatem jest dana o wartości 1.

Przykład

```
#include <stdio.h>
#include <signal.h>
main ( )

{
  int Name ( );
  ssignal (13, Name);
  gsignal (13);
}
  int Name ( )
{
  printf ("Izabela");
  gsignal (13);
}
```

Wykonanie programu powoduje wyprowadzenie napisu Izabela. Dzięki temu, że przed wywołaniem funkcji *printf* zostanie niejawnie wykonana instrukcja:

```
ssignal (13, SIG_DFL);
```

nie dojdzie do nie kończącego się wyprowadzania wymienionego napisu.

Odmiennie od sygnałów, które polegają na związaniu pewnej akcji z podprogramem zamkniętym – funkcją sygnałową – działają powroty. W ich wypadku związanie akcji następuje bowiem z podprogramem otwartym, w szczególności obszernym fragmentem programu. Podobnie jak sygnały, powroty mogą być wykorzystywane do oprogramowania sytuacji wyjątkowych.

Posługiwanie się powrotami umożliwiają dwie funkcje: *setjmp* i *longjmp*, obie wraz z typem *jmp_buf* zadeklarowane w pliku nagłówkowym *setjmp.h*.

Deklaracje prototypowe funkcji *setjmp* i *longjmp* mają postać:

```
int setjmp (jmp_buf env);
void longjmp (jmp_buf env, int val);
```

Wykonanie funkcji *setjmp* powoduje zapamiętanie stanu procesu, który ją wywołał. Do tego celu służy zmienna *env*. Rezultatem funkcji jest dana o wartości 0. Wykonanie funkcji *longjmp* powoduje odtworzenie stanu procesu na podstawie zmiennej *env* i tym samym kontynuowanie wykonywania procesu wywołującego tę funkcję od miejsca wywołania funkcji *setjmp*, ale w taki sposób, jakby rezultatem funkcji *setjmp* była dana o wartości *val*.

Przykład

```
#include <stdio.h>
#include <setjmp.h>

#define TRUE 1
main ( )
{
  FILE * Inp;
  jmp_buf area;
  char Chr;
  Inp = fopen ("SOURCE.DOC", "r");
  if (setjmp (area))
    return;
  while (TRUE){
  Chr = getc (Inp);
  if (feof (Inp))
    longjmp (area, 1);
  putc (Chr, stdout);
  }
}
```

Wykonanie programu powoduje skopiowanie do standardowego pliku wyjściowego, zawartości pliku *SOURCE.DOC*. W chwili rozpoznania, że plik reprezentowany przez *Inp* znajduje się w pozycji końcowej, następuje powrót do miejsca po wywołaniu funkcji *setjmp*, a więc do instrukcji *return*.

NETBIOS

dokończenie ze s. 6

Polecenie wykorzystuje do adresowania numeru własnej nazwy (pole NCB_NUM). Po otrzymaniu datagramu sprawdza specyfikację nazwy i jeśli odpowiada ona numerowi własnej nazwy, to wiadomość jest odbierana. Gdy NCB_NUM = FFH, to wtedy odbierane są wszystkie datagramy przeznaczone dla dowolnej nazwy zadeklarowanej w węźle. Po pomyślnym wykonaniu polecenia udostępniany jest numer nazwy użytkownika odbierającego (pole NCB_NUM). Jeśli polecenie RECEIVE DATAGRAM nie będzie wydane przed użyciem polecenia SEND DATAGRAM, to dane będą stracone. Nie ma możliwości przerwania polecenia wskutek przeterminowania.

RECEIVE BROADCAST DATAGRAM

Odbiera wiadomość od każdego, kto wydał polecenie SEND BROADCAST DATAGRAM. Nie ma możliwości przerwania wykonywania polecenia wskutek przeterminowania.

Przegląd sieci lokalnych dla IBM PC

W ciągu ostatnich lat na rynek wprowadzono i zastosowano kilkadziesiąt lokalnych sieci komputerowych – LAN (ang. *local area network*) dla komputerów osobistych. Kilkanaście z nich jest w zasadzie przeznaczonych dla IBM PC. Żadna z nich nie jest jednak w pełni kompatybilna z pozostałymi. Obecnie większość firm oferujących sprzęt mikrokomputerowy dostarcza już sieci lokalne dla IBM PC. W tabeli wymieniono większość tych sieci, które mogą współpracować także z komputerami kompatybilnymi z IBM PC.

SIECI ORYGINALNE

Część sieci lokalnych skonstruowano specjalnie bądź głównie z myślą o komputerach IBM PC. Do tej grupy należy pięć sieci omówionych poniżej.

3Com EtherSeries

Firma 3Com oferuje wiele wyrobów sieci Ethernet, a jednym z nich jest linia EtherSeries. EtherSeries jest kompletnym systemem Ethernet dla IBM PC. Urządzeniem obsługującym pliki może być zwykły komputer PC/XT lub specjalne urządzenie wykonane przez firmę 3Com. Komputery PC wymagają pakietu programów EtherShare oraz sprzęgu EtherLink tej samej firmy. Alternatywą jest stacja robocza Altos 586 oparta na szybkim procesorze 8086.

EtherShare zarządza wirtualnymi dyskami na poziomie woluminów. Woluminy mogą być prywatne (właścicielem jest określony użytkownik), publiczne (do dyspozycji wielu użytkowników), dostępne tylko do odczytu, dostępne do odczytu i zapisu przez wielu użytkowników. W celu zabezpieczenia przed niewłaściwym dostępem programy użytkowe muszą wykorzystywać semafory i blokady, co umożliwia EtherShare.

EtherShare zapewnia też inne usługi, jak wirtualny spooler drukarki (EtherPrint) lub poczta elektroniczna (EtherMail). EtherMail umożliwia działanie poczty dla wszystkich użytkowników w systemie. Ustanawia dla nich skrytki w obrębie usługodawcy (ang. *file server*) i może odszukiwać skrytki innych użytkowników korzystających z innego oprogramowania EtherShare tej samej sieci. EtherShare jest więc zespołem środków programowych, które można wykorzystywać do wielu celów.

Pakiet programów IBM Cluster

Program IBM Cluster zaprojektowano z myślą o tanim systemie do wprowadzania danych (nazwa „program" jest nieco nieadekwatna, gdyż „The Cluster Control Program" jest całym systemem, a nie samym tylko oprogramowaniem). Porównując ten system z bardziej dojrzałymi i wyrafinowanymi systemami, które są dzisiaj dostępne, można go traktować jako prymitywny.

IBM Cluster działa ze skromną szybkością 375 kb/s posiłkując się procedurą CSMA/CA (CA, ang. collision avoidance, oznacza, iż w systemie są zabezpieczenia pozwalające uniknąć kolizji). Kablem łączącym sieć jest 75-omowy przewód koncentryczny. Praca w tym systemie jest możliwa, gdy główna magistrala jest nie dłuższa niż 1000 metrów (typową długością jest 300 m). Struktura adresowania pozwala dołączyć maksymalnie 64 użytkowników, lecz w praktyce liczba ta jest znacznie mniejsza i zależy od zastosowania. W sieci jest możliwa wymiana informacji pomiędzy poszczególnymi stacjami dyskowymi, można także przesyłać komunikaty i pliki między użytkownikami. Pakiet zawiera podsystem diagnostyczny pomagający identyfikować przyczyny niesprawności.

Sieć IBM PC Network

W połowie sierpnia 1984 roku pojawiła się pierwsza z prawdziwego zdarzenia sieć firmy IBM nazwana IBM PC Network. Jest ona oparta na technologii rozwiniętej przez firmę Sytek Inc. Ta sama firma dostarcza też pakiet adaptera sieci i tzw. jednostkę translatora sieci (ang. *network translator unit*). Jednostka translatora dzięki specjalnemu złączu pozwala połączyć osiem komputerów PC w promieniu sześćdziesięciu kilku metrów. Okablowanie złożone z segmentów, repetytory (wzmacniacze) oraz dodatkowe urządzenia umożliwiają zwiększenie liczby zainstalowanych w sieci komputerów do 72. Sieć ta jest w stanie przesyłać informację z szybkością do 2 Mb/s jednym kanałem w systemie o promieniu 300 m. Zapewnia szereg bardzo niezawodnych usług, a ponadto ma dobrze rozwinięty system samotestowania i diagnostyki.

Pakiet adaptera dla sieci korzysta z układu Intel 82586. Blok pamięci RAM 16K jest obsługiwany przez mikroprocesor Intel 80188, współpracujący z zegarem 6 MHz; blok ten jest przeznaczony na bufory danych, z których korzystają procedury przetwarzające oraz inne programy spełniające funkcje kontrolne. Specjalny, zrobiony na zamówienie, układ VLSI jest używany do obsługi procedur poziomu 5. sesji, według nomenklatury ISO. Istnieje możliwość zorganizowania 32 aktywnych sesji.

Sieć PCnet firmy Orchid

Sieć PCnet wykorzystuje procedury CSMA. Węzły sieci są połączone kablem koncentrycznym. Całość jest bardzo dobrze oprogramowana, m. in. ma moduł wykrywania i arbitrażu kolizji. Ponieważ

Przegląd i porównanie cech lokalnych sieci dla IBM PC

Producent	Sieć	Kabel	Szybkość transmisji (b/s)	Liczba węzłów	Protokół łącza	Maksymalna długość (km)	Obsługa urządzeń	System operacyjny
AST Research	PCnet II	P	800 k	64 K	CS	3,3	D,DR	MS-DOS
Corvus	Omninet	P	1 M	64	CS	3,3	D	MS-DOS, 1,2
Davong	Multilink	K	2,5 M	255	A	7,3	D,DR	MS-DOS, 1,2
Fox Research	10-Net	P	1 M	255	CS		PL, DR, 32	MS-DOS
IBM	Cluster	K	375 k	64	CS	3,3	D	MS-DOS
IBM	PC Network	K	2 M	64	CS	0,7	PL, DR, D, B	MS-DOS
Nestar	PLAN 4000	K	2,5 M	64 K	A	7,3	DR, D, 32, B	MS-DOS, 2
Novell	NetWare/X	K	1,43 M	256	CS	3,3	PL, DR	MS-DOS
Novell	NetWare/S	P	500 k	24	W	0,7	PL, DR	MS-DOS, 1
Orchid	PCnet	K	880 k	64 K	CS	3,3	DR, D	MS-DOS
3Com Corp.	Ethernet	K,S	10 M	64 K	E	3,3	DR, D	MS-DOS
Ungermann-Bass	Net/One PC Baseband	K,S	10 M	64 K	E	3,3	B, DR, B, 32	MS-DOS
Ungermann-Bass	Net/One PC Broadband	P	5 M	64 K	CS	7,3	B, DR, D, 32	MS-DOS

Oznaczenia: P – para skrętna, K – kabel współosiowy, S – światłowód, D – dysk, PL – plik, DR – drukarka, B – brama, 32 – terminal 3270, A – Arcnet, E – Ethernet, CS – CSMA, W – własny, 1 – CP/M-86, 2 – p-System

pakiet sprzęgu sieci służy jedynie do fizycznego połączenia komputera PC z siecią i nie jest wyposażony w procesor, który mógłby sterować funkcjami kanałów przesyłania informacji, ładować bufory, rozstrzygać kolizje itp., te funkcje musi spełniać procesor komputera.

PCnet pracuje z efektywną szybkością 880 kb/s. System połączony jest 75-omowymi kablami koncentrycznymi na odległość 1–2 km, w zależności od typu kabla. Konfiguracja sieci PCnet może być bardzo różnorodna. Dostępny jest wielofunkcyjny pakiet pamięci, różnorodne porty, jak również kanał, przez który można załadować system operacyjny z sieci, co eliminuje potrzebę posiadania dysków w miejscu pracy. Jedną z bardziej interesujących cech PCnet jest możliwość wykonywania poleceń na odległość, np. można wykonać kompilację programu na maszynie oddalonej, pracującej w sieci. Należy być jednak ostrożnym, gdy chce się korzystać ze współdzielenia zasobów, ponieważ nie istnieje firmowa implementacja semaforów, która blokowałaby dostęp do urządzeń. Takie programy piszą sami użytkownicy.

Net/One Personal Connection firmy Ungerman-Bass

Net/One Personal Connection dla komputerów IBM PC wprowadzono pod koniec 1983 roku. Jest to sieć Ethernet z prawdziwego zdarzenia, wykorzystująca procedury wysokiego poziomu Xerox Network Systems (XNS). Ta sieć jako pierwsza dla komputerów PC oferowała tak szeroki zakres usług: obsługę plików, obsługę drukarek a także bramę do SNA (ang. *systems network architecture*). Pakiet sprzęgu sieci nosi nazwę NIU (ang. *network interface unit*) i zawiera układ sterownika VLSI Ethernet produkcji firmy Intel, a ponadto: mikroprocesor 80186, 8 KB pamięci ROM i 128 KB pamięci RAM. Pakiet wymaga jednak zewnętrznego nadajnika-odbiornika sieci Ethernet (produkt firmy 3Com ma takie urządzenie wbudowane).

Pakiet sprzęgu jest więc sam w sobie mikrokomputerem ukierunkowanym na obsługę funkcji kanału i procedur XNS. Jako wyposażenie sieci jest on znacznie lepszy niż oddzielny IBM PC. Net/One PC pracuje z szybkością 5 Mb/s dzięki pakietowi modemu dla częstotliwości radiowych. Mieści się on w obudowie IBM PC i jest połączony z pakietem wersji podstawowej.

Firma Ungermann-Bass zawarła porozumienia z firmą Davong na bardzo potrzebne oprogramowanie sieciowe o nazwie Multilink (współpracujące także z IBM PC). Umożliwia ono także korzystanie z systemu operacyjnego CP/M oraz UCSD p-System na równi z systemem MS-DOS.

POŁĄCZENIA Z KOMPUTERAMI INNYCH FIRM

Inną kategorię sieci stanowią te, które są zdolne działać pod nadzorem różnych systemów operacyjnych, a w ich skład mogą wchodzić nawet minikomputery różnych typów. Różnią się one od wszystkich sieci omówionych powyżej, pracujących pod nadzorem systemu operacyjnego MS-DOS.

Omninet firmy Corvus

Sieć Omninet wykorzystuje procedurę CSMA/PA (PA, ang. *positive acknowledgement* – potwierdzenie pozytywne). Dlatego też może w pełni zabezpieczyć przed przekłamaniem pakiet informacji przeznaczony do transmisji (oczekuje na potwierdzający sygnał zwrotny z węzła odbiorczego). Taki cykl pracy zasadniczo poprawia funkcjonowanie sieci.

Żadna z sieci nie współpracuje z tak liczną grupą mikrokomputerów, jak Omninet. Istnieją jej wersje dla IBM PC, Apple II, DEC Rainbow czy system S-100. Ograniczeniem sieci Omninet są maksymalnie dopuszczalne 63 węzły. Szybkość przesyłania informacji wynosi 1 Mb/s, po linii utworzonej ze skrętki dwuprzewodowej. Połączenie kablowe nie może być dłuższe niż 300 m (bez wzmacniaczy pośrednich) lub 1200 m – ze wzmacniaczami.

Pakiet sprzęgu sieci zwany transporterem zawiera mikroprocesor MC 6801, który steruje komunikacją, obsługuje żądania DMA oraz przerwania. Omninet współpracuje z urządzeniem obsługi dysków, którym jest pakiet z mikroprocesorem Z-80 i wbudowanym sprzęgiem. Można je także wbudować w 5-calowy dysk Winchester. Corvus jest pierwszą firmą, która zintegrowała sterownik sieci ze stacją dysku (do sieci można dołączyć osiem takich dysków).

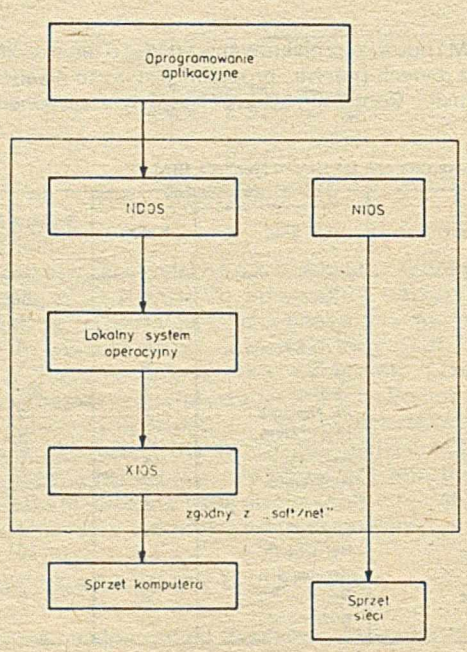
Oprogramowanie systemowe nosi nazwę Constellation i umożliwia współpracę z mikrokomputerami Apple II IBM PC jak i wielu innych typów. Zapewnia ono współpracę z różnymi rodzajami oprogramowania systemowego. Na komputerze Apple można pracować z systemem ApplePascal, DOS 3.3, CP/M, a na IBM PC – DOS 3.0, CP/M-86 czy UCSD p-System. System ochrony jest ograniczony do minimum. Nazwisko użytkownika i hasło wystarczają, aby nawiązać kontakt z siecią. Kontrolę nad dostępem do danych sprawuje specjalny zarządca (ang. *system manager*), który sprawdza głównie prawidłowość korzystania z woluminów. Istnieje też spooler drukarki, który obsługuje dowolne mikrokomputery.

Jednym z bardziej interesujących produktów dla sieci Omninet jest brama do sieci SNA. Zezwala ona na uruchomienie oprogramowania emulatora terminala, jak również ustala wirtualne połączenie z oddalonym właścicielem sieci. Zapewniona jest możliwość jednoczesnego dostępu do bramy przez wielu użytkowników, dzięki programowanemu układowi współpracy z urządzeniami zewnętrznymi.

PLAN 4000 firmy Nestar

Sieć PLAN 4000 jest oparta na procedurze tzw. przekazywania żetonów (ang. *token passing*) powstałej dla sieci Arcnet firmy Datapoint. Na wyższych poziomach współpracuje ona z procedurami XNS. Jest to sieć przesyłająca informacje między węzłami z szybkością 4–5 razy większą niż inne sieci i może być oparta na komputerach IBM PC. Według specyfikacji sieci Arcnet, można przez nią przesyłać dane z szybkością 2,5 MB/s, używając do tego celu kabla koncentrycznego RG-62. Mikrokomputery PC mogą współpracować ze sobą na odległość 10 km. Aktywne wzmacniacze-przekaźniki (ang. *hub*, według terminologii firmy Datapoint), które łączą komputery z siecią muszą występować co 600 m. Sieć PLAN 4000 obsługuje maksymalnie 254 stanowiska.

Urządzenie obsługi plików może wykonywać bardzo złożone funkcje. Jest oparte na mikroprocesorach Motorola 68000 i ma architekturę zgodną z magistralą Multibus. Ma wbudowany streamer, dysk o pojemności ponad 500 MB i port modemu do przeprowadzania diagnostyki odległych punktów sieci. Zarówno na poziomie skorowidzów jak i woluminów obowiązują hasła dostępu (urządzenie obsługi plików posługuje się drzewiastą strukturą skorowidzów). PLAN 4000 może pracować z systemem operacyjnym PC-DOS 3.0 oraz UCSD p-System. Dla komputerów Apple II lub III dostępne są Apple Pascal, DOS 3.3 i CP/M.



Rys. 1. Struktura oprogramowania sieci DRNet

Niepowtarzalnym programem oferowanym przez firmę Nestar jest tzw. serwer bramy, który pozwala na komunikację w czasie rzeczywistym między różnymi sieciami PLAN 4000, przykładowo znajdującymi się na dwóch różnych piętrach budynku. Rzeczywista szybkość przesyłania informacji między sieciami wynosi także 2,5 Mb/s. Mając dwie sieci PLAN 4000, połączone ze sobą i wyposażone w programy obsługi bramy, można założyć wirtualne połączenie z dyskami pracującymi w sąsiedniej sieci i dzięki temu traktować je jak lokalne.

SIECIOWE SYSTEMY OPERACYJNE

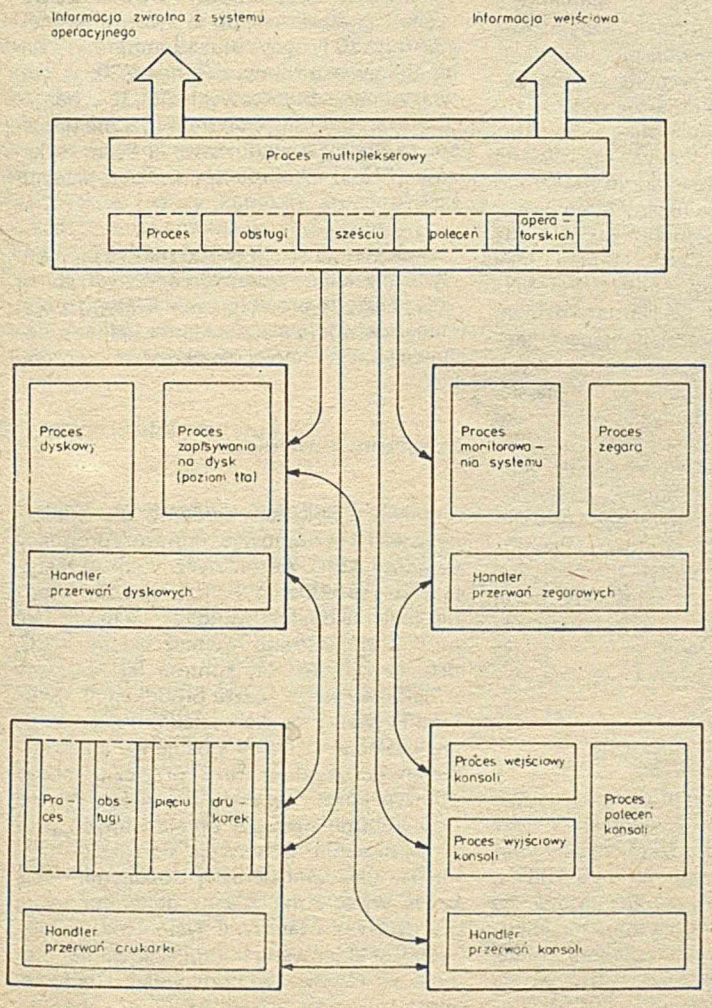
Wiele form wytwarzających oprogramowanie oferuje sieciowe systemy operacyjne, zdolne do współpracy z różnymi sieciami opartymi na komputerach PC. Poniżej omówiono dwa z takich systemów: DRNet firmy Digital Research oraz Net-Ware/OS firmy Novell.

Sieć DRNet firmy Digital Research

DRNet zachowuje zgodność ze wszystkimi systemami operacyjnymi CP/M. Zasadę działania sieci przedstawiono na rys. 1. Moduł NDOS, będący nadbudową systemu operacyjnego, stanowi system obsługi plików. NDOS sprawdza każde przychodzące polecenie i jeżeli jest to polecenie do lokalnego urządzenia peryferyjnego, to zostaje za pomocą lokalnego podsystemu wejścia-wyjścia przekazane do lokalnego systemu operacyjnego. Jeżeli zaś jest to odwołanie do urządzenia wirtualnego, to zostaje wysłane do podsystemu wejścia-wyjścia sieci. Możliwa jest jednoczesna obsługa 16 użytkowników, przy czym każdy z nich może założyć hasło ochrony własnych plików. Z uwagi na fakt, że dzielony dostęp do plików i blokada rekordów są wbudowane w system operacyjny, programy aplikacyjne nie muszą korzystać z semaforów, aby możliwa była praca wielоużytkowa.

NetWare/OS firmy Novell

Oprogramowanie systemu operacyjnego NetWare/OS jest dostępne dla wielu różnych sieci lokalnych, np. Ethernet, PLAN 2000, Omninet, PCnet, a także własnej sieci firmy Novell. We wszystkich tych systemach urządzeniem obsługi plików jest PC/XT lub AT. Wyjątek stanowi NetWare/S, własna sieć firmy Novell, w której urządzenie to oparto na mikroprocesorze Motorola 68000. Aby funkcjonowanie sieci było jak najbardziej skuteczne, wiele funkcji jest realizowanych właśnie przez usługodawcę plików.



Wieloprocesorowy system operacyjny firmy Novell

Oprogramowanie sieci NetWare stanowi nadbudowę systemu DOS, podobnie jak w sieci DRNet. Jego podstawą jest wieloprocesorowy system operacyjny, który „widzi" PC/XT jako urządzenie obsługi plików (rys. 2). Wiele procesów jest wykonywanych równocześnie, a proces obsługi urządzenia zapewnia obsługę aż sześciu oczekujących poleceń. Dostępny jest m. in. proces zapisu i odczytu dysku, zarządzający organizacją niszy. Proces ten zwalnia bloki dyskowe, które były ostatnio zmodyfikowane. Spooler jest procesem na poziomie tła i może obsłużyć trzy drukarki. Ponadto istnieje proces-handler konsoli, obsługujący szeroki zakres poleceń wysyłanych bezpośrednio z klawiatury. Ponieważ system operacyjny jest wyposażony w oprogramowanie obsługi plików, dostęp do plików może być automatycznie blokowany w chwili, gdy któryś z użytkowników otworzył dany plik i dokonuje w nim operacji. Istnieje też możliwość wysyłania komunikatów do innych użytkowników pracujących w sieci.

Obecnie stosowane sieci LAN dla komputerów IBM PC nie są wzajemnie kompatybilne. Po dłuższym okresie ich użytkowania wielu użytkowników może dojść do wniosku, że sieci tanie w chwili instalowania są bardzo kosztowne w eksploatacji, jeśli wziąć pod uwagę stratę czasu spowodowaną długim oczekiwaniem na odpowiedź, gdy w systemie pracuje wiele terminali. W porównaniu z rozwiązaniem opartym ma dużych komputerach, sieci lokalne nie są wyposażone m. in. w takie urządzenia, jak: rozliczanie kont użytkowników, analiza efektywności funkcjonowania systemu, programy administrowania systemem.

Oprac. P. S.

A T M A N

ZAKŁAD PROJEKTOWO WDROŻENIOWY

Jednostka gospodarki uspołecznionej

04-082 WARSZAWA UL.KRYPSKA 39

TEL. 13-25-61 TLX.812530 agro pl

OFERUJE:

* mikrokomputery zgodne z IBM PC/XT/AT w dowolnych konfiguracjach;
* urządzenia peryferyjne (drukarki,digitizery,plotery,streamery i inne);
* lokalne sieci komputerowe (instalacja u użytkownika);
* materiały eksploatacyjne (dyskietki, kasety do drukarek i streamerów,pisaków do ploterów i inne).

PROJEKTUJE I INSTALUJE:

* systemy informatyczne wspomagania zarządzania i produkcji;
* dołączanie urządzeń nietypowych do IBM PC/XT/AT (urządzeń taśmy papierowej, elektr.maszyn do pisania,nietypowych streamerów,skanerów,przetworników A/C,C/A i innych);
* polskie litery.

ZAPEWNIA:

-bezpłatne doradztwo w zakresie doboru sprzętu i oprogramowania;
-bezpłatne oprogramowanie systemowe i narzędziowe;
-12-miesięczny serwis gwarancyjny;
-odpłatny serwis pogwarancyjny;
-dostawy w terminie 7-14 dni.

TEL.13-25-61 TLX.812530 AGRO PL

EO/869/88

Uwaga! ATMAN gwarantuje 18-miesięczny serwis gwarancyjny.

Perspektywy rozwoju układów ASIC

Rozwój specjalizowanych układów scalonych następuje w szybszym tempie, niż wskazywały na to uprzednie analizy rynku. W drugiej połowie 1987 roku [1] oceniano, że w ciągu następnych trzech-czterech lat będą one stanowić ponad połowę sprzedawanych wówczas układów logicznych, których wartość wyniesie 15 mld dolarów (w roku 1986 udział ten wynosił 32%).

Zapowiada to walkę konkurencyjną między dużymi firmami półprzewodnikowymi, które widząc szybki rozwój układów ASIC będą chciały ustalić tu swą pozycję, a przedsiębiorstwami, które pierwsze zaczęły dostarczać układy wytwarzane według częściowej specyfikacji odbiorcy (ang. *Semi Custom*) i które będą starały się utrzymać osiągniętą pozycję na rynku obwodów scalonych. Duże firmy mają większe możliwości wytwórcze, a także rozbudowaną sieć ośrodków projektowych, serwisu i sprzedaży. Przygotowują się one do wejścia na rynek ASIC przez stworzenie obszernych bibliotek komórek standardowych, opartych na aktualnie produkowanych przez nie układach (włącznie z mikroprocesorami 32-bitowymi).

Firmy amerykańskie mają nadzieję stać się konkurencyjnymi w stosunku do firm zagranicznych, zwłaszcza japońskich, zajmujących czołowe pozycje w dziedzinie matryc bramkowych. Na początku lat osiemdziesiątych, próba wejścia firm amerykańskich na rynek ASIC zakończyła się niepowodzeniem. Obecnie najbardziej znane firmy, jak Motorola, Texas Instruments i National Semiconductor solidnie przygotowują się do podjęcia ponownej próby. Natomiast firmy takie jak LSI Logic Corp. i VLSI Technology Inc., związane wyłącznie z tym rynkiem, uważają, że tylko one potrafią zapewnić usługi potrzebne ich klientom. Chodzi tu głównie o możliwości emulacji dowolnych układów logicznych oraz realizacji całych systemów w jednej lub kilku kostkach. Przede wszystkim jednak mają one bezpośredni kontakt z nabywcami przy określaniu założeń, projektowaniu i wytwarzaniu układów oraz szkoleniu personelu. Duże firmy będą musiały iść w ich ślady.

Według prognoz opracowanych przez Dataquest Inc. sprzedaż matryc bramkowych w latach 1987–1992 będzie wzrastać o około 25% rocznie, poczynając od 2,4 mld dolarów w roku 1987 do 7,9 mld w 1992. Dla układów opartych na komórkach standardowych tempo to będzie wynosić ponad 30% (odpowiednio 0,9 i 3,7 mld dolarów). Obecnie większość układów ASIC to matryce bramkowe sprzedawane w dużych ilościach nielicznym wytwórcom komputerów. W przyszłości nabywcami tych układów będą wytwórcy różnego sprzętu pragnący wykorzystać zalety ASIC do poprawienia parametrów swoich wyrobów. Prognozy wydawnictwa „Technical Insight Inc." mówią o znacznym zwiększeniu liczby rozpoczętych projektów ASIC – z 6 tysięcy w 1985 roku do 90 tysięcy w roku 1990. Obejmuje to zarówno dostawców systemów i sprzętu, jak i firmy półprzewodnikowe.

Matryce bramkowe i matryce komórek standardowych

W dziedzinie matryc bramkowych największym dostawcą w 1986 roku była japońska firma Fujitsu (ponad 300 milionów dolarów sprzedaży), zaopatrująca duże firmy amerykańskie z Amdahl na czele. Z analizy rynku wynika jednak, że nie rozwija się ona tak dynamicznie, jak konkurenci. Drugie miejsce (niespełna 200 milionów dolarów sprzedaży) zajmuje LSI Logic, która ze względu na silną konkurencję wytwarza głównie duże układy ASIC obejmujące od 50 do 200 tysięcy bramek, a także kompilatory sprzętowe i układy do zastosowań wojskowych. Na trzecim miejscu (ponad 150 mln dolarów) jest japońska firma NEC specjalizująca się w najnowszych technologiach, takich jak BIMOS o dużej liczbie bramek (obecnie ponad 10 tysięcy). Na dalszych miejscach znajdują się: Toshiba (około 100 milionów dolarów), która wytwarza też matryce dla LSI Logic, Ferranti, Motorola i Texas Instruments. Ta ostatnia zajmowała pierwsze miejsce w dziedzinie matryc komórek standardowych (prawie 100 mln dolarów w 1987 r.). Specjalizuje się ona w mikrofalowych układach do systemów wojskowych opartych na arsenku galu (np. wzmacniacze TGA8300 i 8014), które zastępują układy hybrydowe. Na drugim miejscu są tu połączone firmy General Electric i Radio Corporation of America (ponad 70 milionów), a dalej VTI, Silicon Systems, NCR, Zymos, Harris i Toshiba.

Jednym z czynników zmiejszających liczbę wytwarzanych układów standardowych jest wzrost stopnia ich scalenia. Ocenia się, że wzrósł on średnio z 10 tysięcy na kostkę w 1984 roku do 50 tysięcy w roku 1986. Firmy półprzewodnikowe widzą więc ratunek w układach ASIC oceniając, że na początku lat dziewięćdziesiątych będą one stanowić trzecią część ich produkcji. Przygotowuje się produkcję matryc bramkowych w technologii CMOS oraz najnowszy rodzaj układów ASIC – układy oparte na komórkach opatentowanych (ang. *proprietary*). Są to zwłaszcza mikroprocesory i mikrosterowniki.

Porozumienia między firmami

Przy opracowywaniu i produkcji układów ASIC poszczególne firmy zawierają między sobą różnego rodzaju porozumienia ułatwiające im wejście na rynek. W lipcu 1987 r. Intel i Texas Instruments zawarły umowę, na podstawie której ta ostatnia firma może korzystać z procesorowych komórek Intela. Jedną z pierwszych był układ UC51 – wersja popularnego mikrosterownika 80C51. Podobny układ (MPU 6805) oferuje Motorola. Inny przykład to sprzedaż – przez konsorcjum GE/RCA – układów VGT10 i VGT100 firmy VLSI Technology. Przykładem współpracy dostawcy z klientem jest natomiast porozumienie podpisane w 1986 r. między National Semiconductor a Xerox Corp., zgodnie z którym pierwsza firma przejęła pewne agendy drugiej w zamian za prawo do wykorzystania części opracowań National. Firmy te współpracują też przy określaniu i projektowaniu nowych komórek do biblioteki CMOS.

Ankieta przeprowadzona przez firmę Intel wśród nabywców układów ASIC wykazała, że głównym kryterium oceny układów jest możliwość natychmiastowego uruchomienia prototypu (65% ankietowanych) oraz czas opracowywania układów (62%). Takie czynniki, jak wysoka wydajność obliczeniowa, koszt jednostkowy lub zwartość były uważane za mniej istotne cechy (nieco poniżej 50%), a koszt opracowania brało pod uwagę tylko 28% ankietowanych.

Układy ASIC stosowane w komputerach umożliwiają skrócenie czasu opracowania nowych wyrobów, poprawienie ich parametrów roboczych i niezawodnościowych, a także zmniejszenie liczby pakietów przez zwiększenie funkcjonalności kostek. Przykładem takich rozwiązań może być rodzina mikrosuperkomputerów C1 firmy Convex Computer Corp. (matryce CMOS, które początkowo zawierały po 8 tysięcy bramek, a później 20 tysięcy) oraz skomputeryzowane stanowiska robocze Sun-4/260, w których z matryc bramkowych CMOS o bardzo dużym stopniu scalenia zrealizowano procesor centralny o zredukowanej liście rozkazów (RISC) i koprocesor zmiennoprzecinkowy.

Brak standaryzacji wśród matryc bramkowych zmusza do stosowania różnych sprzęgów i narzędzi projektowych. Dlatego projektanci decydują się zwykle na jednego dostawcę, przynajmniej dla określonego projektu.

Problemy kontroli

Innym problemem zarówno dla odbiorców, jak i wytwórców jest kontrola produkcji układów ASIC. Automatyczny sprzęt testujący do układów o bardzo wielkim stopniu scalenia dostosowano tak, by mogły być na nim sprawdzane również układy ASIC, przy czym koszt tej kontroli jest znaczny – 30–50% całego kosztu produkcji, tj. dwukrotnie więcej niż dla kostek o dużym stopniu scalenia. Obecnie stosuje się specjalne testery do układów ASIC, przy czym chcąc obniżyć koszt, dąży się do wykorzystania przy kontroli narzędzi opracowanych przy projektowaniu. Obserwuje się rozbieżności między układami bardziej złożonymi, gdzie każde wyjście ma własny generator testu, a prostszymi i tańszymi systemami, w których jest stosowane multipleksowanie. Próbuje się również wbudowywać układy samosterujące w kostki ASIC, co znacznie upraszcza kontrolę i obniża ich koszty.

Projektanci stosujący układy ASIC, w zależności od wymagań szybkości, mocy lub celowości użycia układów analogowych, wybierają odpowiednią technologię, właściwy format tych układów i wreszcie charakter współpracy z dostawcą. Zastosowanie tych układów wiąże się z pewnym ryzykiem zarówno przy projektowaniu, jak i przy realizacji. Najwięcej obaw budzi projektowanie, przy czym za szczególnie trudną uznaje się sytuację, gdy nie można zrealizować modelu z układów standardowych i trzeba oprzeć się wyłącznie na symulacji. Korzysta się wówczas z uprzednio zaprojektowanych bloków lub specjalnych systemów obliczeniowych (np. Genesil firmy Silicon Compiler Systems).

Również rozwiązanie konstrukcyjne układu ma znaczny wpływ na jego parametry, łącznie z ceną i niezawodnością. Na przykład, stosowanie połączeń spawanych (TAB – ang. *tape-automated bonding*) ujednolica impedancję wyjść.

Początkowo poszczególne rozwiązania układów ASIC były ściśle rozgraniczone i na przykład przy dużych zamówieniach, powyżej 25 tysięcy układów rocznie, były stosowane układy komórek standardowych. Obecnie rozróżnienia te zatarły się i opracowuje się matryce bramkowe o złożonych funkcjach, nawet przy ilościach sięgających 100 tysięcy rocznie. Powstają wreszcie rozwiązania łączące obie te grupy urządzeń, a wytwarzanie matryc bezkanałowych (ang. *sea of gates*) pozwala realizować funkcje osiągane dawniej tylko w bibliotece komórek. Jednocześnie rozwój programowych narzędzi projektowania umożliwił osiągnięcie układów o bardzo dużej liczbie bramek. Współczesne układy komórek standardowych stanowią kombinację elementów pamięciowych, matryc programowanych i bloków innych funkcji specjalnych. Proces wytwarzania tych układów obejmuje co najmniej 12-16 etapów maskowania (przy matrycach bramkowych było tylko 4-6 etapów).

Narzędzia projektowania

Narzędzia do projektowania układów ASIC mogą być uniwersalne (są to wówczas skomputeryzowane stanowiska robocze wspomagające biblioteki wytwórców) albo też specjalizowane, opracowane przez poszczególne wytwórnie systemy projektowania, czasami oparte na mikrokomputerze (jak Gateaid Plus/PC firmy Matra-Harris w cenie zaledwie 795 dolarów), a czasami rozbudowane stanowiska kilkadziesiąt razy droższe. Stosuje się je do symulacji, przy czym wraz z rozwojem technologii (przewidywano 120 tysięcy użytecznych bramek średnio w kostce w 1988 r.) zagadnienia te stają się coraz bardziej złożone. Użytkownik spotyka się często z problemami różnych standardów i formatów tych narzędzi. Poszczególne firmy starają się stworzyć w miarę obszerne systemy, jak Modular Design Environment firmy LSI Logic, które nie są jednak uniwersalne.

Przykładem narzędzia, w którym wykorzystano ostatnie osiągnięcia sztucznej inteligencji, jest Design Adviser firmy NCR Microelectronics. Zapewnia ono możliwość określenia takich parametrów jak opóźnienie i synchronizacja, a także możliwość kontroli oraz wprowadzanie i wyprowadzanie danych. Te ostatnie zagadnienia, zwłaszcza wprowadzanie schematów elektrycznych, jest istotnym podzespołem, czasami opcjonalnym, każdego systemu. Jeśli biblioteki ASIC zawierają odpowiednie oznaczenia schematowe, to można używać uniwersalnych programów wprowadzania. Symulację przeprowadza zwykle projektant, mając w ten sposób lepszą kontrolę nad procesem projektowania.

Istnieją również symulatory uniwersalne i firmowe, które są zwykle dokładniejsze, jeśli chodzi o przebiegi w czasie. Czasami są to specjalne programy do obliczeń opóźnień (np. Vita firmy NCR). Natomiast symulatory uniwersalne umożliwiają dalszą symulację na poziomie pakietu czy systemu. Dalsza grupa narzędzi służy do rozmieszczania elementów na płytce krzemowej. Mogą to być tak zwane kompilatory krzemowe wytwarzające takie rozmieszczenia, a czasami też modele symulacyjne.

Rozwój matryc bramkowych

Na zakończenie warto omówić oferowane w połowie 1987 roku układy ASIC i narzędzia ich projektowania [2]. W dziedzinie matryc bramkowych oferowano 108 wyrobów 37 firm. Najwięcej typów oferowały firmy Fujitsu (9) i Motorola (7). Przeważały układy wykonane w technologii CMOS (ponad 70%); niespełna 12% to układy bipolarne, ponad 5% – układy ECL i niecałe 3% – układy z arsenku galu. Standardowe wymiary (szerokość ścieżek, odstępy między nimi) zawierają się w granicach od 1 do 6 μm, przy czym trzecia część wyrobów opiera się na technice 2 μm, a prawie czwarta część – 1,5 μm. Całkowita liczba bramek w układzie waha się średnio od 1450 do 13 860, przy czym najmniejsza liczba wynosi 52, a największa 129 tysięcy. W stosunku do ilości całkowitej wykorzystuje się średnio 82-84% bramek. Częstotliwość przerzutników realizowanych z użyciem tych układów waha się od 30 kHz do 2,1 GHz. Opóźnienie na obciążonym układzie zawiera się w granicach od 70 ps do 50 ns. Ponad 90% układów jest wyposażonych w narzędzia projektowe, m. in. modele symulacyjne i narzędzia umożliwiające wczytywanie schematów, symulację układów logicznych oraz analizę czasową. Narzędzia do rozmieszczania elementów ma ponad 60% układów. Wśród systemów, na których może być realizowane projektowanie, najczęściej wymienia się Mentor i Daisy (dla prawie 80% układów) oraz VAX i Valid (około 40%).

Jeśli chodzi o typ obudowy to najwięcej jest PGA (*Pin Grid Array* – matryca kołkowa) i DIP (*Dual in Line Package* – obudowa o dwóch rzędach wyprowadzeń) około 70% oraz PLCC (*Plastic Leaded Chip Carries* – plastikowe obudowy kostek z doprowadzeniami) 50% i LCC (*Leadless Ceramic Chip* – ceramiczna kostka bezodprowadzeniowa) 36%. Szacunkowe koszty opracowania układu zawierają się między 5 a 125 tysiącami dolarów. Jako tak zwane drugie źródło oferowanych matryc bramkowych występuje 18 firm, najczęściej LSI Logic (8 razy) i Motorola (5 razy). Drugiego źródła nie ma 44% ofert.

Rozwój komórek standardowych

Podobne dane dla układów komórek standardowych przedstawiają się następująco: oferowano 71 układów 39 firm, przy czym trzech producentów (California Micro Devices, GE/RCA i VTC) dysponowało czterema różnymi układami. Tu również przeważały układy CMOS (84%, w tym 54% typu Sigate), bipolarnych było 7%, a opartych na związkach galu 5%. Wymiary standardowe zawierają się w granicach od 0,8 do 8 μm. Częstotliwość przerzutników waha się od 5 MHz do 3 GHz. Opóźnienia wynoszą od 70 ps do 6 ns.

Jeśli chodzi o narzędzia projektowania, to sytuacja jest podobna jak przy matrycach bramkowych. Najczęściej używane tu systemy to Mentor i Daisy (prawie 80%) oraz VAX (54%) i Valid (41%). Przeważają obudowy DIP (95% układów), a prawie 80% układów występuje w obudowie PGA i PLCC. Ponad 94% układów jest wyposażonych w modele symulacyjne oraz w narzędzia umożliwiające wczytywanie schematów, symulację układów logicznych i analizę czasową, a 56% w narzędzia do rozmieszczania elementów. Koszty opracowania są ponad półtora raza wyższe w stosunku do matryc bramkowych i zawierają się w granicach od 18 do 150 tysięcy dolarów. Jako drugie źródło układów komórek standardowych występowało również 18 firm, najczęściej NCR (7 razy) i VTI (5); 41% oferowanych układów nie miało drugiego źródła.

Rozwój narzędzi

Do projektowania układów ASIC 36 firm oferowało 45 narzędzi symulacyjnych, przy czym największy asortyment miały: Motorola (4 narzędzia) i Thomson Semiconducteurs z Francji (3). Można z nich korzystać na 45 różnych komputerach, skomputeryzowanych stanowiskach roboczych lub rodzinach komputerów, wśród których najczęściej wymieniane było stanowisko firmy Apollo (23 razy) i VAX firmy DEC (22 razy); różne systemy IBM występują 32 razy. Wśród stosowanych systemów operacyjnych najczęściej występują Unix (36 razy) i VMS (22 razy) oraz Aegis (18 razy). Jeśli chodzi o formaty wprowadzania, to najczęściej występują tu systemy firmy Daisy, Mentor i Valid. Analiza jest przeprowadzana najczęściej na poziomie bramek (w 40 przypadkach), następnie zachowania się układu (25) i przełączania (23). Szybkość symulacji wa-

dokończenie na s. 43

Rynek drukarek komputerowych w 1988 roku

Najszybciej rozwijającymi się rodzajami drukarek są laserowe i mozaikowe drukarki uderzeniowe o 24 igłach. Wzrasta też produkcja drukarek kolorowych. Natomiast coraz mniej sprzedaje się drukarek czcionkowych, zwłaszcza margeritkowych.

Drukarki 24-igłowe stanowią najbardziej zaawansowaną grupę drukarek uderzeniowych (tab. 1). Według Data Corp. w 1991 roku stanowić one będą 36% wszystkich drukarek uderzeniowych mozaikowych. Dają one dobrą jakość wydruku, a szybkość druku jest zbliżona do 200 znaków na sekundę. Posiadają też dostatecznie dużą pamięć buforową, pozwalającą na szybką transmisję. Cena ich, natomiast, choć wyższa od ceny konwencjonalnych drukarek mozaikowych, jest niższa od ceny drukarek laserowych. Zaletą ich jest również niski koszt eksploatacji (znacznie mniejszy, aniżeli dla drukarek laserowych) i możliwość wydruku kolorowego.

Oprócz drukarek wymienionych w tabeli 1 należałoby jeszcze wspomnieć o 24-igłowej drukarce 35/5X firmy Toshiba (cena 1500 dolarów) i MSP50 firmy Citizen, ekonomicznej drukarce 9-igłowej o szybkości wydruku 300 zn/s.

Drukarki laserowe charakteryzują się przede wszystkim bardzo dobrą jakością wydruku i dlatego stanowią podstawowy zespół drukujący w systemach poligraficznych wspomaganych komputerowo.

Podstawowymi parametrami drukarek laserowych są: szybkość wydruku oraz kompatybilność z określonym sprzętem. Cena jest na ogół konsekwencją wyboru tych parametrów. Inne czynniki wpływające na wybór techniki laserowej, to jej wysoka rozdzielczość (na ogół, 300 punktów na cal); możliwość wydruku na różnych formatach papieru i liczba

Tabela 1. Drukarki mozaikowe (18–24-igłowe)

Producent	Model	Szybkość wydruku (zn/s)		Liczba igieł	Rozdzielczość (zn/s)	Sprzęg i emulacja	Uwagi
		Jakość wydruku QL lub zbliżona	Maksymalna				
ALPS	Q 200	80	240	18/24	240×216	RS232, Centronics, Epson	Możliwość wydruku kolorowego
ALPS	Q 2100	80	400	18/24	60×240	Centronics, RS232	Kompatybilna z DEC
C.ITOH	9815	160	333	24	360×180	IBM Proprinter, Toshiba P 351, Qume Sprint II	Możliwość wydruku kolorowego, cena 1995 dol.
EPSON	LQ 500	50	180	24	360×180	Centronics, RS232	Cena 499 dol.
GENICOM	Seria 1000	100	200	18	240×144	RS232, Centronics	Możliwość wydruku kolorowego
OKIDAT	Microline 393	120	450	24	360×360	Epson LQ	Możliwość wydruku kolorowego
SEIKOSHA	SL 130 AI	60	180	24	Wysoka rozdzielność	Równoległy Epson LQ 100, grafika IBM	Cena 999 dol.
SEIKOSHA	SBP-10	–	800	18	–	PC, Równoległy	Cena 4495 dol.
STAR MICRONICS	NB 24-10/24-15	72	216	24	–	Równoległy PC, Epson	Cena 749–999 dol., zestawy czcionek w wymiennych kasetach
STAR MICRONICS	NB-15	100	300	24	–	Równoległy PC, Epson	Cena 1399 dol.
TEXAS INSTRUMENTS	875/877	80	240	24	–	Epson LQ 1500, Centronics	Cena 799–999 dol.

Tabela 2. Drukarki laserowe

Producent	Model	Szybkość (str/min)	Rozdzielczość (pkt/cal)	Sprzęg	Emulacja	Uwagi
C.ITOH	Lips10+	10	300	Równoległy, szeregowy	Epson Diablo, HP Laserjet	Pamięć 512 KB do 2 MB, cena 2995 do 3995 dol.
FUJITSU	RX 7100	5	300	–	HP Laserjet+Diablo, Epson IBM Proprinter, HPGL	Cena 1160 dol.
GENICOM	5010	10	300	Centronics, RS232	Diablo, grafika IBM, HP Laserjet	Zespół drukujący Canon-3 cena 495–3695 dol. kompatybilne z PC/XT/AT
OKIDATA	Laserline6	6	300	Centronics, RS232	HP Laserjet+Diablo, IBM, Qume, Nec, Apple	Cena podstawowa niższa od 2000 dol.
OLYMPIA	Laserstart6	6	300	Centronics, RS232	Diablo, HP, IBM (opcja), Epson (opcja)	1,5 MB, cena 2399 dol.
QMS	810	8	300	RS232, Appletalk, równoległy	HP Laserjet, HPGL, Diablo	Język drukowania – Postscript
QUADRAM	Qualaser Postscript	8	300	Centronics, Appletalk, RS232	Apple	Język drukowania – Postscript
QUME	Srip Ten	10	300	Appletalk, RS232, Centronics	HP Laserjet+	1,3 MB, możliwość sterownika, Postscript
RICOH	Laser600	6	300	Centronics, RS232	Epson, IBM Proprinter, HP Laserjet+	Kompatybilna z PC/XT/AT
STAR MICRONICS	Laser Printer 8	8	300	Centronics RS232	Epson, Diablo, IBM Proprinter, HP Laserjet+	1 MB, 4 emulacje standardowe
TEXAS INSTRUMENTS	22016	15	300	Centronics, Appletalk, RS232	Diablo, Epson, Qume, HP Laserjet+	Zestawy czcionek, Postscript w kasetach, cena 4595 dol.
TEXAS INSTRUMENTS	Omnilaser 2106	6	300	Mac-IBM	–	Sterownik Postscript

Tabela 3. Drukarki kolorowe

Producent	Model	Sposób wydruku	Rodzielczość	Szybkość (zn/s)	Sprzęg i emulacje	Uwagi
ALPS	P 2400C	18–24 igły	60×240	12/360	Centronics, RS232, Epson	Oprogramowanie kompatybilne z PC, XT, AT
BROTHER	M-4018	18 igieł	–	480	Równoległy, szeregowy Epson, IBM	Cena 1695 dol., do dyspozycji podajnik arkuszy
GENICOM	3310	9 igieł	72×144	75/360	Diablo, grafika IBM, RS232, Centronics	Do dyspozycji podajnik arkuszy
HONEYWELL BULL	4/40 i 4/41	igły	–	70/300	Epson JX 80, IBM Proprinter	Cena 1195 i 1395 dol.
XEROX	4020	strumieniowy	120×120	20/40	RS232, Centronics	Poprawa rozdzielczości do 240×120

pojemników z papierem, które można wykorzystać. Droższe drukarki laserowe (w cenie 5500-7000 dolarów) mają szybkość wydruku 8-10 stron na minutę i stosują język drukowania Postscript firmy Adobe lub Xpress firmy Xerox. Tańsze (1800–2700 dolarów) mają szybkość rzędu 6 stron na minutę, a średnie (2700-4500 dolarów), przy szybkości 8 stron na minutę, wykorzystują języki drukowania Quic firmy QMS lub PDL albo DDL firmy Hewlett-Packard.

Jak widać z tabeli 2, większość firm zajmujących się drukarkami oferuje dziś drukarki laserowe. Istnieje duże zainteresowanie możliwością wydruku barwnego przy użyciu tych drukarek, lecz przy obecnym stanie techniki jest on bardzo drogi. Wydruk kolorowy znany jest już od kilku lat, lecz sprzedaż sprzętu na szerszą skalę rozpoczęła się dopiero teraz, gdy opracowano lepsze i tańsze technologie. Drukarki kolorowe są realizowane w trzech technikach: igłowej, strumieniowej i termicznej.

Realizacja uderzeniowa jest najprostsza i najbardziej zaawansowana, a tym samym najtańsza. Coraz więcej drukarek mozaikowych uderzeniowych posiada możliwość wydruku kolorowego, który uzyskuje się uderzeniem przez kolorową taśmę, a poszczególne barwy powstają przez nałożenie kolorów podstawowych (purpurowy, niebieskozielony (fr. *cyan*), żółty i czarny), co umożliwia otrzymywanie siedmiu kolorów łącznie z czarnym.

Technika strumieniowa z kolei zapewnia szerszą paletę kolorów, zwykle od 36 do 125 odcieni. Uzyskuje się to przez dobieranie proporcji atramentu w czterech podstawowych barwach.

Technika przenoszenia termicznego jest dość koszowna, lecz zapewnia wysoką jakość. Można tu uzyskać 256 odcieni, a czasami nawet więcej.

W tabeli 3 wymieniono najpopularniejsze drukarki kolorowe.

Na wystawie Comdex'87 pokazano wiele drukarek kolorowych specjalizowanych, pracujących głównie w systemach poligraficznych. Należałoby tu wymienić drukarki firmy QMS ze sterownikiem języka Postscript. Jest to wysokiej klasy drukarka termiczna, oparta na zespole drukującym firmy Mitsubishi i zawierająca pamięć operacyjną o pojemności 12 MB (kosztuje ok. 9 tysięcy dolarów). Rozdzielczość wynosi 300 elementów na cal, a więc jest taka sama jak dla drukarek laserowych. Zapowiedziano szybkie opracowanie narzędzi programowych, opartych na znanych systemach jak Autocad i Windows.

Podobnym przykładem jest nieco droższa drukarka firmy Tektronix, która dołączona do komputera Macintosh II, zapewnia paletę 16,7 miliona kolorów, przy takiej samej rozdzielczości i pojemności pamięci, co drukarka QMS. Rozpowszechnienie się tego typu drukarek doprowadzi niewątpliwie do obniżenia ich ceny.

Jeśli chodzi o kolorowe drukarki strumieniowe, to oprócz podanej w tabeli 3 drukarki firmy Xerox warto wspomieć o modelu FP 510 firmy Canon, który przy rozdzielczości 160 elementów na cal umożliwia uzyskanie 256 odcieni, przy 64 różnych poziomach. Taki wydruk uzyskiwany przy małych wymiarach (nieco większych od wizytówki), natomiast przy dwóch poziomach można drukować na arkuszach A4. Również firma Howtek wzbogaciła swą ofertę o nowe drukarki tego typu, o rozdzielczości 240 punktów na cal pionowo i 480 punktów poziomo, przy ponad 250 tysiącach różnych możliwości. Uzyskuje się to dzięki specjalnie opracowanemu atramentowi stałemu, dającemu doskonały wydruk.

Na zakończenie, warto wymienić laserowy system drukujący Risograph 3500 firmy Riso Kagaln, który można określić jako „kolorowany", gdyż po wydruku każdego koloru należy ręcznie zmieniać system nadawania farby, przy czym przy każdej takiej zmianie automatycznie czyszczony jest bęben. Sprzężenie operatorskie jest tu typu wizyjnego. System ten jest interesujący tylko przy masowym wydruku, gdyż jego szybkość drukowania wynosi 120 stron na minutę.

Opracował **J. RYŻKO**
na podstawie „Minis et Micros", nr 293-294

Perspektywy rozwoju układów ASIC

dokończenie ze s. 41

hała się od tysiąca do 5 milionów bramek na sekundę. Parametr ten był inaczej określany w zdarzeniach na sekundę i wówczas zakres zmian zawierał się w granicach od 500 zdarzeń do 75 milionów zdarzeń na sekundę. Ceny wahały się od 3 do 120 tysięcy dolarów, a za oprogramowanie – od 495 dolarów do 150 tysięcy.

Narzędzia do rozmieszczania elementów

Ponadto, 23 firmy oferowały 30 narzędzi umożliwiających rozmieszczanie elementów (w tym Thomson Semiconducteurs – 3). Najczęściej stosowanym komputerem był VAX (15 razy), a następnie stacje firmy Apollo (9). W 14 narzędziach był wykorzystywany system operacyjny VMS, a w 8 Aegis i Unix. Najczęściej spotykanym formatem wprowadzania jest system EDIF (7 razy), a następnie systemy Mentor, Daisy i Silver-Lisco (po 6 razy). Narzędzia te dla matryc bramkowych najczęściej są dostosowane do technologii CMOS (16 razy) i ECL (5 razy), przy czym maksymalna liczba bramek waha się od 3 do 125 tysięcy, a w ramach różnych technologii najmniejsza liczba występuje średnio dla układów BiMOS (4,5 tysiąca), a największa dla matryc bezkanałowych (ang. *sea of gates*) – 72,5 tysiąca. Odpowiednio dla komórek standardowych najpopularniejsza jest także technologia CMOS (18 razy), następnie układy arsenku galu (5 razy) i bipolarne (4 razy). Występuje tu większa rozpiętość maksymalnej liczby komórek – od 20 sztuk – dla filtrów funkcji analogowych – do 500 tysięcy. Istnieją też narzędzia o nieograniczonej liczbie komórek. Narzędzia te są dostosowane do 37 bibliotek różnych firm; najczęściej występują tu biblioteki Motoroli (4 razy) oraz NEC i Mostek (po 3 razy). Ceny oprogramowania tych narzędzi wahają się od 15 do 80 tysięcy dolarów.

Opracował
JAN RYŻKO

LITERATURA

[1] Special Issue – The ASIC Takeover – It's Coming Faster. Electronics No. 16, pp. 57, 1987
[2] Special Report on ASICs and ASIC Design Tools. Computer Design, No. 15, pp. 41, 1987

Program bibliotekarza Ady

Jednym z głównych problemów wynikających z realizacji i pielęgnacji oprogramowania średniej i dużej wielkości jest zarządzanie wszystkimi jego składnikami: tekstem źródłowym poszczególnych modułów, modułami wynikowymi po kompilacji, ładowalnymi postaciami binarnymi oraz dokumentacją całego pakietu i poszczególnych modułów, przy założeniu, że każdy moduł może występować w wielu wersjach. W związku z tym często podczas kompilacji konieczne jest posiadanie kilku wersji jednego modułu (np. wersji oficjalnej, poprzedniej oraz aktualnie uruchamianej). Dotyczy to zarówno programów szczególnie ważnych, jak również realizowanych przez duży zespół programistów. W celu umożliwienia eksploatacji oprogramowania w różnych konfiguracjach, często konieczne jest posiadanie różnych programów dostosowanych do poszczególnych konfiguracji. Może również istnieć wiele wersji odpowiadających kolejnym fazom doskonalenia programu.

Użytkownik powinien mieć możliwość korzystania nie tylko ze swoich modułów, ale także z innych należących do tego samego projektu. Aby uniknąć strat czasu przy wykonywaniu tego rodzaju operacji, konieczne jest utworzenie ścisłych powiązań między modułami.

Język Ada może zapewnić wymaganą spójność programów, a opracowany w firmie Alsys program bibliotekarza umożliwia osiągnięcie maksymalnej elastyczności przy zachowaniu reguł obowiązujących w Adzie dla rozłącznej kompilacji modułów. Równoległe tworzenie bibliotekarza i kompilatora pozwoliło na właściwe powiązanie tych programów między sobą, zapewniając w ten sposób automatyczną kontrolę spójności i poprawności każdego kompilowanego modułu.

Rozłączna kompilacja w Adzie

Ada narzuca ścisłe ograniczenia na rozłącznie kompilowane moduły. Z drugiej strony jednak ten język jest elastyczny, dając możliwość programowania zstępującego przy użyciu oddzielnych podjednostek, wstępującego przy użyciu pakietów lub połączenia tych dwu metod.

Rozłączna kompilacja w Adzie opiera się na koncepcji biblioteki, która zawiera wszystkie niezbędne informacje o modułach tworzących realizowany program. Podczas kompilacji modułu wszystkie moduły, od których jest on zależny, powinny być już wcześniej skomplikowane w tej samej bibliotece.

Elastyczność programowania w Adzie jest wzmacniana przez rozróżnienie specyfikacji i ciała pakietu. Specyfikacja zawiera deklaracje obiektów widocznych dla modułów użytkownika. Obiektami tymi mogą być typy, stałe, zmienne, podprogramy lub zadania. Ciało pakietu zawiera algorytmy wszystkich tych podprogramów i zadań oraz ewentualnie część inicjalizacji. Dzięki takiemu rozróżnieniu moduł użytkownika może być kompilowany, gdy tylko zostaną skompilowane specyfikacje pakietów przez niego używanych. Ponadto kompilacja ciała pakietu nie wymaga kompilacji modułów korzystających z tego pakietu. Również kompilacja oddzielnej podjednostki nie pociąga konieczności powtórnej kompilacji wszystkich korzystających z niej modułów, wymaga jednak powtórnej kompilacji wszystkich jej podjednostek w celu zachowania spójności programu. Z tego samego powodu w wypadku powtórnej kompilacji specyfikacji pa-

```
-- Interakcyjna sesja Piotra
* ada PK_SPECIF L1                    -- kompilacja pliku źródłowego
  NO ERROR                            -- zawierającego specyfikację
* ada PK_BODY L1                      -- pakietu PK w bibliotece L1
  NO ERROR                            -- i kompilacja ciała PK w L1
* ada PK_S1_BODY L1                   -- kompilacja podjednostki S1
  NO ERROR                            -- pakietu PK w bibliotece L1
* ada PK_S2_BODY L1                   -- kompilacja podjednostki S2
  NO ERROR
* library_manager L1                  -- wywołanie bibliotekarza dla L1
  COMMAND:
* list »                              -- pytanie o zawartość L1
  Name: PK
  package specification  compiled SEP 09, 1988, 10:43
  Sons: PK package body
  Name: PK
  package body           compiled SEP 09, 1988, 10:48
  Sons: PK.S1 subprogram body
        PK.S2 subprogram body
  Father: PK package specif
  Name: PK.S1
  subprogram body        compiled SEP 09, 1988, 10:51
  Father: PK package body
  Name: PK.S2
  subprogram body        compiled SEp 09, 1988, 10:54
  Father: PK package body
  COMMAND:
* quit                                -- zakończenie pracy bibliotek.
-- Paweł chce użyć pakietu PK Piotra do kompilacji swojego modułu A
-- w bibliotece L2.  Poniżej przedstawiono jego sesję.
* library_manager L2                  -- wywołanie programu biblio-
  COMMAND                             -- tekarza dla biblioteki L2
* link PK/ps of L1                    -- dowiązanie dla specyfikacji
  COMMAND:                            -- pakietu PK między L1 i L2
* list »                              -- pytanie o zawartość L2
  Name: PK
  package specification  compiled SEP 09, 1988, 10:43
  From: L1                            -- dowiązanie z biblioteki L1
  Sons: K package body
  Name: PK
  package body           not compiled
  Father: PK package specif
  COMMAND:
* quit
* ada A_SPECIF L2                     -- kompilacja spec. modułu A
  NO ERROR                            -- zawierającego "with PK;"
-- Piotr chce wykonać powtórną kompilację specyfikacji PK.
* ada PK_SPECIF L1
  NO ERROR
  WARNING: units of L1 to br recompiled are
           package body      PK
           subprogram body   PK.S1
           subprogram body   PK.S2
-- Paweł chce wykonać kompilację ciała swojego modułu A
* ada A_BODY L2
  ERROR: link of PK from L2 to L1 is invalid,
         probably because PK was recompiled in L1
  NO CODE PRODUCED
-- Paweł musi dokonać pewnych zmian w swojej bibliotece.
* library_manager L2
  COMMAND:
* delete_invalid_links
  Bad link found of PK to L1 -> deleted
  WARNING: units of L2 to be recompiled are:
           package specif      A
  COMMAND:
* link PK/ps of L1
  COMMAND:
* quit
* ada A_SPECIF L2                     -- teraz kompilacja specyf. i
  NO ERROR                            -- i ciała A jest już możliwa
* ada A_BODY L2
  NO ERROR
-- W celu ułatwienia uruchamiania programu Paweł chce wykonać kompi-
-- lację swojej wersji PK.S1 w bibliotece L2. Tworzy postać
-- źródłową, którą nazywa PK_S1_BODY_PAWEL.  Aby kompilacja PK.S1 w
-- L2 była możliwa, musi istnieć dowiązanie do przodka PK.S1, tj. do
-- ciała PK.
* library_manager L2                  -- tworzenie w bibliotece L2
  COMMAND:                            -- dowiązania ciała PK do bibl.
* link PK/pb of L1                    -- L1; PK.S1 i PK.S2 są rozpoz-
  COMMAND:                            -- nawane jak skompilowane w L2
* quit
* ada PK_S1_BODY_PAWEL                -- możliwa jest teraz
  NO ERROR                            -- kompilacja PK.S1 w L2
-- Po wykonaniu kompilacji ciała podprogramu ROOT zawierającego
-- klauzulę "with A;", Paweł chce utworzyć moduł ładowalny. Używa
-- do tego celu programu wiążącego.
* bind ROOT in L2
     Module names        kind          fetched in
          ROOT           sb              L2
          A              ps              L2
          A              pb              L2
          PK             ps              L1
          PK             pb              L1
          PK.S1          sb              L2
          PK.S2          sb              L1
  Load module ROOT_LM created.
-- Moduł PK.S1 został pobrany z L2 i jest zależny od wersji two-
-- rzonej przez Pawła.  Moduł PK.S2 nie był skompilowany w L2, w
-- został automatycznie pobrany z L1, ponieważ PK znajduje się w
-- W ten sposób utworzony moduł ładowalny jest całkowicie spójny
-- zgodnie z zasadami przyjętymi w Adzie.
```

kietu wszystkie korzystające z niej moduły muszą zostać powtórnie skompilowane.

Przedstawione zasady wystarczają, aby zagwarantować spójność programu realizowanego przez jedną osobę, przy istnieniu tylko jednej wersji każdego modułu. Ada nie rozwiązuje jednak problemu istnienia wielu wersji modułów i nie zapewnia pomocy w wypadku realizacji programów przez liczne zespoły programistów.

W środowisku programowym Ady konieczna jest więc obecność programu bibliotekarza, który będzie kontrolował relacje między modułami, a także relacje wynikające z organizacji wytwarzania i pielęgnowania programu.

Rozszerzenie pojęcia biblioteki

Pojęcie biblioteki istnieje już w samej Adzie. W omawianym przedsięwzięciu zaproponowano jego rozszerzenie w dwóch kierunkach.

Po pierwsze, biblioteka zawiera informacje o modułach tworzących program. Element biblioteki jest znany, gdy był kompilowany lub gdy jest podjednostką modułu, który był kompilowany. Rozszerzenie polega na zwiększeniu ilości informacji o modułach ponad wymagania kompilatora. Informacje zawierające, na przykład, tekst źródłowy modułu lub opis poddrzew, których wierzchołkiem jest dany moduł, mogą być następnie wykorzystywane przez różne programy użytkowe ze środowiska kompilatora, a także przez sam program bibliotekarza.

Drugie rozszerzenie polega na umożliwieniu kompilatorowi i innym programom z jego środowiska – dostęp do wielu bibliotek. Koncepcja biblioteki w Adzie zakłada bardzo ograniczoną liczbę jej użytkowników, w szczególności – tylko jednego. W wypadku dużego projektu często jeden z programistów ma kontrolę nad oddzielnie kompilowanymi poddrzewami, których poprzedni poziom w drzewie lub elementy wskazywane klauzulami **with** (tworzącymi sprzężenie) są opracowywane przez kogoś innego. Wprowadzone rozszerzenie ma pozwolić na uniknięcie konieczności powtórnej kompilacji sprzężenia osobno przez każdego członka pracującego nad danym projektem, co zwiększa prawdopodobieństwo błędów i niespójność programu.

Kompilator, aby mógł wykonać kompilację modułu **A**, musi znać sprzężenie tego modułu. W tym celu wprowadzono nowy typ obiektu – tzw. dowiązanie (ang. *binding, link*). Dowiązanie jest przez kompilator rozpoznawane jako deskryptor modułu, więc kompilator „widzi" moduł w bibliotece użytkownika, podczas gdy w rzeczywistości znajduje się on w innej bibliotece. Dowiązania są tworzone w sposób kontrolowany w celu zachowania reguł języka Ada i zgodności modułów. Tworzenie dowiązań z określonym modułem umożliwia użytkownikowi odwoływanie się do tego modułu z innych modułów przez klauzule **with**, jak również tworzenie swojej własnej wersji podjednostki tego modułu. W wypadku powtórnej kompilacji modułu, do którego tworzone są dowiązania, użytkownik otrzymuje informację, że dowiązania te są nieaktualne.

Z możliwości posiadania wielu wersji jednego modułu w różnych bibliotekach wynika konieczność istnienia narzędzia, które umożliwiłoby wytworzenie modułu ładowalnego. Tę funkcję ma spełniać program wiążący (ang. *binder*). Użytkownik podaje nazwę programu głównego i biblioteki, w której on się znajduje, a program wiążący przeszukuje całą bibliotekę znajdując odpowiednie moduły, dzięki wiązaniom wprowadzonym przez kompilator oraz dowiązaniom utworzonym przez użytkownika.

Implementacja

Program bibliotekarza zawiera cztery części:

1. Pakiet Ady spełniający wszystkie podstawowe funkcje bibliotekarza. Pakiet ten jest używany przez narzędzia programowe ze środowiska kompilatora w celu uzyskania dostępu do funkcji bibliotekarza, np. dostępu do bibliotek, wprowadzania modułów (nowych lub w czasie wymiany), tworzenia dowiązań itp.
2. Interakcyjny program pozwalający użytkownikowi na polecenie z terminala wykonania określonych funkcji pakietu i otrzymanie komunikatów o błędach.

Narzędzie to dostarcza również informacji o stanie biblioteki. W ten sposób możliwe jest otrzymanie listy modułów bibliotecznych, dla których konieczna jest powtórna kompilacja w wypadku zmiany określonego modułu. Użytkownik ma również dostęp do tekstu źródłowego modułu i może go wydrukować lub skopiować do swojego pliku.

3. Program wiążący.
4. Interakcyjny program dostarczający programom zarządzającym projektem środków do tworzenia, niszczenia i incjowania bibliotek.

Program ten umożliwia również korzystanie z operacji bibliotekarza, ale jest to zarezerwowane dla osób uprawnionych.

Wykorzystanie bibliotekarza

Pierwsza wersja programu bibliotekarza i programu wiążącego została wykorzystana przy opracowywaniu kompilatora Ady i jego środowiska programowego dla trzech różnych komputerów. W każdym z nich programy te zarządzały blisko czterdziestoma wzajemnie powiązanymi bibliotekami i ponad 4000 modułów i dowiązań, całkowicie spełniając stawiane im wymagania.

Druga wersja bibliotekarza i programu wiążącego, także przeznaczona do współpracy z kompilatorami firmy Alsys, została pozbawiona pewnych ograniczeń zawartych w wersji poprzedniej. Nową wersję tych programów przystosowano do współpracy z innymi programami usługowymi ze środowiska programowego kompilatora, np. narzędziami automatycznej rekompilacji, programem uruchomieniowym itp. W tym celu oba programy (bibliotekarz i program wiążący) musiałby przechowywać dodatkowe informacje dla każdego modułu.

Jako przykład wykorzystania programu bibliotekarza prześledzimy tworzenie programu przez dwóch użytkowników. Przyjmiemy, że biblioteki **L1** i **L2** są używane odpowiednio przez Piotra i Pawła. Przed rozpoczęciem pracy **L1** i **L2** są puste. W przykładzie linie rozpoczynające się gwiazdką (*) oznaczają informacje wprowadzane przez użytkowników z klawiatury, a linie nie zawierające gwiazdki – komunikaty systemu. Znaki „—" oznaczają komentarze przyjęte w języku Ada. W przykładzie nazwy poleceń zostały podane w całości, w praktyce mogą być skracane. Skróty oznaczają: **ps** – package specification, **pb** – package body, **sb** – subprogram body. Zapis „**PK/ps**" oznacza specyfikację pakietu PK.

Przewidywane są pewne rozszerzenia programu bibliotekarza wypływające bezpośrednio z doświadczenia w pracy z tym programem.

Obecnie używany program bibliotekarza dopuszcza istnienie w danej chwili tylko jednej wersji danego modułu w określonej bibliotece, przy czym w innych bibliotekach mogą znajdować się inne wersje tego modułu. Powoduje to konieczność istnienia dużej liczby bibliotek, ponieważ każdy projekt wymaga pracy z wieloma bibliotekami. Nowa wersja bibliotekarza ma być wolna od tego ograniczenia – ma dopuszczać istnienie wielu wersji jednego modułu w tej samej bibliotece.

Powtórna kompilacja w bibliotece **L1** modułu, do którego istnieją odwołania z biblioteki **L2**, uniemożliwia realizację tych odwołań. W wypadku, gdy użytkownik biblioteki **L2** chce korzystać z aktualnej wersji modułu z **L1** i nie chce być informowany o jego ewentualnej powtórnej kompilacji, zamiast polecenia **LINK** może użyć zlecenia **COPY**. Po wykonaniu tego polecenia dalsza praca ze skopiowanym w ten sposób modułem odbywa się tak, jakby moduł ten był kompilowany w **L2** i nie istniały dla niego żadne odwołania do **L1**. Wykonanie polecenia **COPY** odbywa się dużo szybciej niż powtórna kompilacja postaci źródłowej modułu w **L2**.

Program bibliotekarza w dużym stopniu rozwiązuje problemy występujące w realizacji dużych projektów w języku Ada. Kompilatory Ady firmy Alsys są przystosowane do pracy w środowisku zawierającym wiele bibliotek i współpraca z bibliotekarzem nie wymaga żadnych dodatkowych zmian. Zastosowanie programu bibliotekarza pozwala na dużą oszczędność czasu w opracowywaniu programów.

Opracowała
DOROTA INKIELMAN

Powyższy artykuł był planowany do umieszczenia w nr 10, 1988, lecz nie mógł się tam ukazać z powodu braku miejsca. (przyp. red.)

Środowisko programowe Ady

Potrzebę tworzenia środowisk programowych zaczęto odczuwać na początku lat siedemdziesiątych, kiedy to pisanie oprogramowania przestało być sztuką zarezerwowaną dla kilku specjalistów, a zaczęło stawać się przemysłem mającym swoich konstruktorów, pracowników i klientów. Wkrótce okazało się, że tworzenie i pielęgnowanie oprogramowania wymaga opracowania odpowiednich metod postępowania, dostosowanych do poszczególnych faz pracy nad oprogramowaniem. W przemyśle oprogramowania, jak w każdym młodym przemyśle, konieczne jest istnienie odpowiednich narzędzi wspomagających każdy z etapów produkcji. Postęp technologiczny w dziedzinie sprzętu i systemów komputerowych ułatwił i przyspieszył pojawienie się wielu narzędzi programowych: edytorów tekstowych, pakietów uruchomieniowych, programów zarządzających bibliotekami, baz danych itp.

Drugi etap rozwoju pracowni programistycznych rozpoczął się kilka lat później, gdy ewolucja języków programowania doprowadziła do powstania idei połączenia wszystkich rozproszonych programów usługowych w jednolity system, umożliwiając ich wzajemną współpracę, przenośność i rozbudowę przez dołączanie nowych modułów. Ta tendencja znalazła swój wyraz w powstaniu systemu Unix, który umożliwia tworzenie oprogramowania prostego koncepcyjnie, lecz złożonego użytkowo i mającego duże możliwości przetwarzania.

ROZWÓJ ŚRODOWISKA ADY

Środowisko programowe Ady ma zadanie ułatwiać tworzenie oprogramowania w tym języku. Choć już dawno podjęto próby tworzenia takich środowisk, ze względu na złożoność i nowość napotykanych problemów, na pełne rozwiązanie tego zadania trzeba będzie jeszcze poczekać. Poniżej wskazano na istniejące już programy służące jako narzędzia ułatwiające uruchomianie oprogramowania.

Dokument STONEMAN

Powstały na początkowym etapie rozwoju Ady dokument STONEMAN jest zbiorem wymagań określających model łączący w sobie możliwość otrzymania oraz użytkowania efektywnych narzędzi do tworzenia programów. Dąży się w nim do scalenia konwencjonalnych narzędzi programowych i utworzenia zestawu, który pomieściłby w sobie różnorodne metody oraz narzędzia programowania – także nowe, aktualnie niedostępne lub nie używane. Na model ten o nazwie APSE (ang. *Ada Programming Support Environment*) składają się trzy elementy: centralna baza danych, sprzężenie użytkowe systemu oraz zbiór narzędzi programowania.

Baza danych służy do przechowywania informacji związanych z każdym projektem na każdym etapie jego istnienia. Sprzężenie zawiera język sterowania, będący nie tylko sprzężeniem użytkownika, ale również sprzężeniem systemowym dla bazy danych i innych narzędzi programowych. Narzędzia programowe to programy usługowe przeznaczone do opracowywania, pielęgnowania i zarządzania oprogramowaniem.

W celu zapewnienia przenośności, dokument STONEMAN definiuje dwa pojęcia: jądro (ang. *Kernel Ada Programming Support Environment*, KAPSE) i minimalne środowisko (ang. *Minimal Ada Programming Support Environment*, MAPSE). KAPSE zapewnia bazę danych, funkcje komunikacji oraz mechanizmy umożliwiające wykonanie programu aplikacyjnego lub programu należącego do MAPSE. Wykonuje swoje funkcje zwracając się bezpośrednio do sprzętu i systemu operacyjnego. MAPSE dostarcza minimalny zbiór narzędzi programowych napisanych w języku Ada, współpracując wyłącznie z KAPSE i pozwala na opracowywanie i pielęgnowanie programów w Adzie. Pełne środowisko APSE jest skonstruowane przez rozszerzenie MAPSE tak, aby zapewnić lepszą obsługę niektórych zastosowań i metod programowania. Taka struktura gwarantuje przenośność – przeniesienie APSE i programów aplikacyjnych na inny komputer wymaga jedynie modyfikacji modułu KAPSE.

Poziomy KAPSE i MAPSE

Głównym składnikiem KAPSE jest baza danych, w której obiekty mają swoje nazwy i mogą mieć dowolny format. W bazie danych zapamiętane są relacje między obiektami. Obiekty mają atrybuty, np. atrybut historii, który dotyczy sposobu ich utworzenia. Każda operacja na obiektach z bazy danych musi zachowywać ten atrybut. Każdy obiekt ma atrybut kategorii, określający typ związanej z nim informacji, oraz atrybut dostępu. Atrybuty obiektów ułatwiają programom usługowym APSE korzystanie z bazy danych.

Baza danych powinna mieć postać umożliwiającą przechowywanie jej w sposób spójny. Powinna również ułatwić programom usługowym korzystanie z jej zasobów, dostarczając informacji o atrybutach i relacjach między obiektami, a także wykonywać funkcje związane z kontrolą dostępu oraz mechanizmami bezpieczeństwa.

KAPSE spełnia następujące funkcje:
- dostarcza elementarnych operacji wejścia-wyjścia,
- zapewnia mechanizmy umożliwiające wzajemne wywoływanie się programów APSE,
- umożliwia użytkownikom wysyłanie poleceń do programów usługowych,
- dostarcza pakietów opisujących główne obiekty bazy danych, jak formy pośrednie powstałe w wyniku kompilacji tekstu źródłowego.

Według dokumentu STONEMAN poziom MAPSE powinien zawierać:
- uogólniony edytor tekstowy przeznaczony do tworzenia dokumentacji i do pisania programów źródłowych,
- specjalny program redagujący, tzw. superprinter, pozwalający w sposób standardowy i czytelny wyświetlać teksty źródłowe i dokumenty,
- kompilator lub interpreter Ady,
- program wiążący, umożliwiający tworzenie postaci ładowalnej programów z modułów skompilowanych,
- skrośny program ładujący,
- procesor odwołań zewnętrznych (ang. *cross references*),
- analizatory statyczne i dynamiczne,
- program administratora plików umożliwiających ich porównywanie i zarządzanie nimi,
- interpreter języka poleceń,
- program konfiguracyjny korzystający z atrybutów przypisanym obiektom należącym do bazy danych,
- symboliczny program uruchomieniowy.

Powyżej poziomu MAPSE mogą istnieć inne programy usługowe, jak na przykład: edytor syntaktyczny, system dokumentacji, system zarządzania projektami, programy pomiarowe, narzędzia specyfikacyjne i koncepcyjne oparte na odpowiednich metodykach itp.

Pierwsze realizacje

Raport STONEMAN był na dużą skalę wykorzystywany przez dwóch kontrahentów Departamentu Obrony USA: Softech dla środowiska ALS (ang. *Ada Language System*) i Intermetrics dla środowiska AIE (ang. *Ada Integrated Environment*). O rozmiarach tych projektów świadczy fakt, że obie firmy opracowały swoje środowiska programowe na podstawie kontraktów opiewających na ok. 10 milionów dolarów. Stopień realizacji obu projektów był daleki od pierwotnych założeń – wydajność systemu ALS była znacznie niższa niż wymagana przez użytkowników, a w projekcie AIE prace koncentrowały się jedynie na realizacji kompilatora. Brak powodzenia obu przedsięwzięć był wynikiem zbyt dużych oczekiwań. Ogrom wymagań spowodował, że obie firmy musiały rozpocząć prace w dziedzinach, w których nie miały dotychczas doświadczenia. Nie było bowiem dotychczas środowiska programowego, umożliwiającego prowadzenie prac programistycznych na tak szeroką skalę.

Prace nad środowiskiem programowym Ady prowadziły również inne firmy: ROLM wspólnie z Data General i DEC.

Zespoły konstruktorów ROLM i Data General przyjęły założenia dokumentu STONE-

MAN, ale postawiły sobie za cel wyposażenie swoich klientów w środowisko języka Ada, wykorzystując 32-bitowy komputer ECLIPSE. Środowisko to wykorzystuje wiele standardowych składników systemu operacyjego: edytory, programy ładujące, systemy plików. Do całości włączono dodatkowo program redagujący, program bibliotekarza umożliwiający rozłączną kompilację modułów, symboliczny program uruchomieniowy i narzędzia konfiguracyjne. Tak zbudowany system, pomimo niespełniania wszystkich założeń raportu STONEMAN, stanowi podstawowy zestaw narzędzi i jest wykorzystywany przez wielu użytkowników do prac nad oprogramowaniem w Adzie.

Wysiłki firmy DEC były skierowane w stronę opracowania wszechstronnego kompilatora Ady i harmonijnego włączenia go do systemu operacyjnego VMS. Konstruktorzy opracowali własny specjalizowany program bibliotekarza, pozwalający na kontrolę powiązań między jednostkami i automatyczną rekompilację niezbędnych modułów. Dodatkowo, standardowy program uruchomieniowy wyposażono w możliwość śledzenia wykonania segmentów związanych z przełączaniem zadań i sytuacjami wyjątkowymi. Dla wszystkich zastosowań, użytkownicy dysponują klasycznym zestawem narzędzi operacyjnego systemu VMS. Warto również wspomnieć o włączeniu Ady do zbioru języków obsługiwanych przez sterowany składnią edytor systemowy LSE (ang. *Language Sensitive Editor*).

PROJEKT ALSYS

Przedsięwzięcie podjęte przez firmę Alsys jest oparte na dokumencie STONEMAN. Przyjęto, że środowisko będzie składać się z jądra KAPSE, zapewniającego współpracę z macierzystym systemem operacyjnym i zbioru programów tworzących poziom APSE, przy czym jądro powinno być jak najmniejsze.

Jądro składa się z dwóch pakietów Ady: jeden zapewnia współpracę z systemem wejścia-wyjścia, a drugi udostępnia odwołania systemowe umożliwiające tworzenie procesów potomnych i komunikację między procesami. Tak zbudowane jądro może być rozszerzone przez dołączanie kolejnych pakietów. Dzięki takiej strukturze jest ono w pełni przenośne i może współpracować z dowolnymi systemami operacyjnymi, w tym również najprostszymi (MS-DOS). Taka minimalizacja jądra spowodowała przeniesienie wszystkich innych funkcji opisanych w raporcie STONEMAN na wyższy poziom struktury.

Poziom MAPSE zawiera wszystkie operacje związane z zarządzaniem bazą danych i współpracą z użytkownikiem. Składa się na to ok. 60 pakietów będących składnikami programów usługowych. Programy te korzystają tylko z tych modułów, które są im niezbędne. Na tym poziomie dostępne są: pakiet zarządzania oknami, pakiet pobierania polecenia, pakiet opisu biblioteki i zawartych w niej obiektów itp.

Dwa podstawowe składniki – bibliotekę i sprzężenie użytkowe – omówiono poniżej.

Biblioteka Ady

Biblioteka Ady jest równoważna bazie danych zdefiniowanej w dokumencie STONEMAN z możliwie najprostszym zbiorem funkcji. Od początku założono, że będzie ona nie tylko umożliwiała rozłączną kompilację modułów Ady, ale również wspomagała pracę innych narzędzi. Taka decyzja spowodowała dodatkową komplikację programu kompilatora, który generuje nie tylko kod wynikowy, ale również bibliotekę informacji wykorzystywanych przez inne narzędzia. Wewnętrzna reprezentacja programu otrzymana przez analizę tekstu źródłowego jest zachowana w postaci umożliwiającej odtworzenie wersji źródłowej wraz z komentarzami.

Biblioteka zapewnia spójność zawartych w niej informacji z odpowiednimi konwencjami języka Ada.

Punkty wejścia do biblioteki są pamiętane w odpowiednich pakietach. Opisy zawartych w niej obiektów wraz z funkcjami dostępu są zawarte w oddzielnych pakietach zgodnych z typami obiektów. Pozwala to programom narzędziowym wybierać tylko te pakiety, które są niezbędne do ich funkcjonowania. Struktura ta usuwa niedogodności występujące w projekcie bazy danych zawartym w raporcie STONEMAN.

Podstawową zaletą takiej koncepcji biblioteki jest występowanie niezależnych podbibliotek połączonych między sobą. Dzięki takiej strukturze w projekcie można korzystać z wielu podbibliotek połączonych ze sobą równorzędnie, co pozwala uniknąć struktury hierarchicznej. Umożliwia to opracowanie modułów korzystających ze wspólnych składników oraz posiadanie wielu wersji jednego modułu dostępnych w jednej bibliotece. Spójność odwołań Ady jest gwarantowana dzięki weryfikacji w czasie łączenia modułów. Na przykład, pierwsza wersja kompilatora Ady, firmy Alsys, została opracowana w systemie posiadającym 45 podbibliotek i blisko 3000 modułów z tysiącami powiązań z elementami wspólnymi.

Sprzężenie użytkowe

Współpraca narzędzi z użytkownikiem jest realizowana przez zbiór pakietów zapewniających zarządzanie wszystkimi aspektami takiej komunikacji. Również w tym wypadku każdy program usługowy korzysta tylko z niezbędnego dla siebie zestawu pakietów. Taki minimalny zbiór pakietów musi zawierać:
- wykaz skrótów,
- samouczek,
- makropolecenia,
- wprowadzanie poleceń z plików,
- prowadzenie dziennika i odtwarzanie historii
- system menu,
- programowanie klawiatury terminali,
- zapamiętywanie wyników,
- zarządzanie oknami.

Właściwe wykorzystanie tego zbioru funkcji pozwala na opracowanie standardowego sposobu współpracy z użytkownikiem, wspólnego dla wszystkich programów, a także na minimalizację kosztów opracowania następnych programów.

Zbiór narzędzi

Środowisko programowe Ady zawiera programy specyficzne dla Ady oraz standardowe programy wspomagające prace nad oprogramowaniem. Do pierwszej kategorii należą:
- symboliczny program uruchomieniowy,
- edytor syntaktyczny,
- dekompilator,
- procesor struktury tekstów źródłowych,
- edytor symboliczny Ady,
- analizator powiązań,
- program profilujący.

Do drugiej kategorii należą programy zarządzania tekstami źródłowymi i programy specyficzne dla różnych metod programowania.

Przedsięwzięcie firmy Alsys skoncentrowało się wokół pierwszej kategorii programów z założeniem maksymalnej ich przenośności. Programy drugiej grupy istnieją już na wielu systemach, ale sposób ich adaptacji do specyficznych potrzeb użytkowników wymaga jeszcze uściślenia. Niektóre z tych programów (np. zarządzające tekstami źródłowymi) są niezależne od Ady i nie wymagają szczególnej adaptacji.

Oprac. **DOROTA INKIELMAN**

Elektrownia
im. Tadeusza Kościuszki
28–230 Połaniec

odsprzeda

2 PAMIĘCI FERRYTOWE (w pełni sprawne technicznie) do zestawu komputerowego R-32 typu FZP-X/36/1Y-256 kB

Zainteresowanych prosimy o kontaktowanie się z Wydziałem Systemów Komputerowych, tel. 50-126 lub 50-179 wewn. 61-94, 68-80.

EO/934/88

Konferencje

Teoretyczne podstawy informatyki

W ubiegłym roku, w dniach 22–26 czerwca, w Kazaniu, ZSRR, odbyła się międzynarodowa konferencja dotycząca teoretycznych podstaw informatyki, pn. „Fundamentals of Computation Theory" – FCT'87. Wzięło w niej udział ok. 180 uczestników z całego świata. Na kilkunastu sesjach wygłoszono 84 referaty i przedstawiono 30 komunikatów (w formie plakatów). Z bardziej znanych informatyków referaty wygłosili m. in. J. L. Jerszow i J. Hartmanis. Całość materiałów konferencyjnych ukazała się w serii „Lecture Notes in Computer Science" wydawnictwa Springer-Verlag.

Jednym z wyników konferencji było opracowanie deklaracji dotyczącej rozwoju teoretycznych podstaw informatyki, którą publikujemy w całości.

Deklaracja uczestników Międzynarodowej Konferencji FCT'87

Wybuch rewolucji informacyjnej, zapoczątkowanej i podtrzymywanej przez rozwój technologii, zaznacza początek nowej ery w historii ludzkości – ery rosnącego wpływu przetwarzania informacji i wiedzy na rozwój społeczeństwa i życie codzienne. Odtąd ogólna szybkość postępu i standard życia będą bardziej zależne od naszej zdolności do efektywnego opracowywania, utrzymywania i użycia zasobów informacji i wiedzy niż – zasobów naturalnych. Nauka jest kluczowym narzędziem w rękach społeczeństwa do przyspieszenia tych procesów.

Nauka który wyłoniła się z cybernetyki matematycznej, teoretycznych podstaw komputerów i pewnych działów matematyki, zwana w skrócie informatyką, jest obszerną i podstawową dziedziną o kluczowym znaczeniu dla społeczeństwa w obszarze rewolucji informacyjnej. Jej głównym celem jest badanie praw i ograniczeń rządzących procesami informacyjnymi i ich realizacjami, w podobny sposób jak fizyka i nauki pokrewne badają prawa i ograniczenia rządzące procesami fizycznymi i ich realizacjami. Jej dodatkowym celem jest opracowywanie nowych i skutecznych narzędzi intelektualnych, niezbędnych do rozwiązywania problemów przetwarzania informacji we wszystkich dziedzinach aktywności ludzkiej i do wspomagania rozwoju metod, technik i systemów wzmacniających potęgę ludzkiego umysłu, gdziekolwiek to potrzebne. Ta nauka będzie miała w przyszłości znaczący wpływ na rozwój innych nauk, na przemysł w ogóle i nowoczesną technologię w szczególności, na rolnictwo, ekonomię, kształcenie itd., a także na poszerzenie granic ludzkiego pojmowania wszechświata i filozoficznego kontekstu, w którym uprawia się naukę.

Nowa rola, jaką ma do odegrania nauka w rozwoju społeczeństwa w ogóle, a w przemyśle i ekonomii w szczególności, wymaga, aby ta nauka reagowała o wiele lepiej i szybciej niż dotychczas na potrzeby społeczeństwa, i poświęcała więcej uwagi zastosowaniom oraz szybszemu wdrażaniu wiedzy naukowej do nowych i ważnych wyrobów. Jednakże bardzo istotne jest uświadomienie sobie, że nie tylko w długich, lecz i w krótkich okresach nauka nie będzie spełniała potrzeb społeczeństwa w sposób pożądany, o ile wszystkie nauki podstawowe nie będą reprezentować bardzo wysokiego poziomu i otrzymywać dostatecznego wsparcia w swoim rozwoju. Obecny nacisk na zastosowania i wdrożenia, spowodowany rosnącym naciskiem na utylitarność nauki, i towarzyszące mu niedocenianie nauk podstawowych, stanowi duże niebezpieczeństwo, że nauka w ogóle już w bliskiej przyszłości nie będzie mogła sprostać wymaganiom społecznym w wystarczającym stopniu. W informatyce teoretycznej jest to szczególnie istotne z następujących powodów:

- Informatyka teoretyczna jest dziedziną bardzo obszerną i bardzo młodą, dlatego jej baza naukowa nie jest jeszcze wystarczająco rozwinięta i zorganizowana. Ta baza jest także zbyt mała w porównaniu z różnorodnością i znaczeniem zastosowań, którym musi służyć. To także powoduje konieczność pogłębienia badań nad podstawowymi pojęciami i technikami informatyki.

- W informatyce związek między teorią a praktyką jest ściślejszy niż w większości innych nauk, a sukces w dziedzinie zastosowań zależy o wiele bardziej od wyników badań teoretycznych. Wyniki teoretyczne są często stosowane niemal bez żadnego opóźnienia. Z tego względu informatyka teoretyczna – w porównaniu z wieloma innymi, starszymi naukami – nie mogła jeszcze uzyskać dość wyników do wykorzystania w przyszłości i wypracować zasobu wiedzy do późniejszego użycia. Z tego też powodu ta nauka nie tylko nie może sobie pozwolić na spowolnienie badań podstawowych, często wymuszane przez nacisk na zastosowania, ale gwałtownie potrzebuje ich znacznego przyspieszenia.

- Nacisk na szybkie zastosowania jest obecnie w informatyce, z wielu powodów, o wiele silniejszy niż w innych naukach. Oczywiście, bardzo ważne jest konstruowanie prototypowych i doświadczalnych systemów sprzętowych i programowych. W żadnym wypadku nie może to jednak zastąpić badań podstawowych, niezmiernie potrzebnych do tego, aby umożliwić konstruowanie lepszych systemów w przyszłości. Jest również prawdą, że w informatyce teoretycznej można – być może łatwiej niż w innych naukach – uzyskać niemal bezpośrednio użyteczne wyniki. Nie jest to jednak w żadnym wypadku powód, aby zakładać, że w tej dziedzinie wystarcza prowadzenie mniejszej liczby badań podstawowych niż w innych naukach – wręcz przeciwnie.

- Informatyka była dotychczas rozważana, rozwijana, wspomagana i oceniana głównie jako nauka stosowana, a nie jako nauka podstawowa, co byłoby pożądane.

Z wymienionych powodów duże znaczenie dla społeczeństwa powinno mieć zbudowanie solidnej bazy naukowej informatyki, co umożliwiłoby skupienie się na badaniu problemów podstawowych i rozwoju całkowicie nowych narzędzi. Aby to osiągnąć, należy jak najszybciej poczynić następujące kroki:

- zapewnić środki dla najważniejszych badań w skali odpowiedniej do ich rozmiaru, stylu i jakości oraz potencjału tej dziedziny,

- zjednoczyć środowisko naukowe i utworzyć ośrodki badawcze, w których skupi się i utrzyma czołówka badaczy,

- stworzyć warunki, aby informatyka teoretyczna mogła być atrakcyjna dla utalentowanych młodych ludzi, najbardziej skutecznych we wprowadzaniu i rozwijaniu całkowicie nowych idei,

- ustanowić lepszy i bardziej efektywny system kształcenia,

- stworzyć warunki, aby dla badań podstawowych w informatyce teoretycznej możliwa była bardziej aktywna i elastyczna współpraca międzynarodowa,

- spowodować, aby czołowi badacze mogli ogłosić i uświadomić społeczeństwu cele, osiągnięcia i potrzeby informatyki.

Kazań, czerwiec 1987

Ogłoszenia • Ogłoszenia

Programy, instrukcje i udoskonalenia techniczne dla komputerów ATARI, AMSTRAD, COMMODORE, IBM oferuje Agencja Komputerowa 41–200 Sosnowiec, P-157, tel. 63-29-35.

EO/423/87

Ogłoszenia • Ogłoszenia

Terminologia systemów operacyjnych (3)

dokończenie z III strony okładki

przyjęto nazywać po angielsku *partition*. Polski odpowiednik tego terminu nie powinien brzmieć partycja, lecz – **strefa**, co już zostało uznane przez niektórych autorów. Obszar pamięci operacyjnej o fragmentach dynamicznie alokowanych i zwracanych nazywa się po angielsku *heap*, co można przetłumaczyć na **stóg**, choć spotkałem też inne określenie – sterta.

Na ogół, wszyscy wiedzą, co oznacza sformułowanie, że program rezyduje w pamięci operacyjnej – jego kod jest załadowany do tej pamięci. Dlatego poprawne są zwroty – **program rezydujący** i **program rezydentny**. Warto jednak dodać, że choć można mówić **polecenie rezydentne** (polecenie, którego kod rezyduje w pamięci), to zwrot polecenie rezydujące jest niepoprawny, ponieważ stwierdzenie, że polecenie rezyduje w pamięci jest pozbawione sensu.

Ważne pojęcie oznaczające zmianę położenia programu w pamięci operacyjnej określa się po angielsku mianem *relocation*. Najwłaściwszym polskim odpowiednikiem tego terminu angielskiego jest wyraz **relokacja**, ponieważ nie można do tego celu użyć ani wyrazu **przemieszczenie** ani **przesunięcie**, gdyż w informatyce znaczą zupełnie co innego, jako odpowiedniki angielskich terminów *displacement* i *shift*. Z podobnych względów należy używać takich terminów pokrewnych, jak **relokować** (ang. *to relocate*), **relokowalny** (ang. *relocatable*) itd.

Innym ważnym terminem dotyczącym pamięci jest angielski przymiotnik *contiguous*, który odnosi się do obszaru zajmującego jeden niepodzielny fizycznie fragment pamięci operacyjnej lub dyskowej. W języku polskim nie można użyć w tym znaczeniu przymiotnika ciągły jak czynią niektórzy autorzy, ponieważ jest on odpowiednikiem terminu angielskiego *continuous* i znaczy co innego. Spośród pozostających do wyboru dwóch innych przymiotników polskich – **spójny** i **zwarty** – proponowałbym wybrać ten drugi, ponieważ w informatyce nie jest tak obciążony innymi znaczeniami.

Ostatni termin, który chcę rozważyć w tym cyklu jest najtrudniejszy. Chodzi o angielski wyraz *checkpointing*, oznaczający czasowe przeniesienie kodu programu (zadania, procesu) – podczas jego wykonywania – z pamięci operacyjnej na dysk, a po zwolnieniu pamięci – przeniesienie go z powrotem w to samo miejsce. Choć sam termin występuje głównie w systemie operacyjnym RSX-11, to pojęcie nie jest ograniczone tylko do tego systemu i występuje pod różnymi nazwami, np. *swapping, rollin/rollout* itp., w innych systemach.

W dotychczasowych publikacjach w języku polskim każdy z autorów używał innego odpowiednika wyrazu *checkpointing*, przy czym żaden z tych odpowiedników nie jest właściwy. Nie można nazwać tej operacji **wywłaszczaniem** (A. Paprocki, P. Wolański), ponieważ ten termin dotyczy raczej procesora i jest polskim odpowiednikiem angielskiego – *pre-emption*. Nie można również użyć wyrazów **usuwanie** i **ściąganie** (J. Deminet), ponieważ **usuwanie** jest odpowiednikiem angielskich terminów *remove* i *delete*, a wyraz **ściąganie** ma zbędne zabarwienie emocjonalne. Podobne zabarwienie emocjonalne ma wyraz **wymiatanie** (M. Maniecki), prawdopodobnie dlatego pozbawiony znaczeń technicznych. Nie można też mówić o **czasowym wstrzymaniu** (J. Wiśniewska), ponieważ termin **wstrzymanie** jest odpowiednikiem angielskiego *hold*, a **punkt wstrzymania** to po angielsku *breakpoint*, a nie *checkpoint*. Sądzę jednak, że niewielka modyfikacja tej ostatniej propozycji – na **powstrzymywanie** – może być do przyjęcia dla wszystkich. Wówczas czasownik to *checkpoint* znaczyłby po polsku **powstrzymywać**, a rzeczownik *checkpoint* – **punkt powstrzymania**.

JANUSZ ZALEWSKI

INSTYTUT ORGANIZACJI PRZEMYSŁU MASZYNOWEGO

uprzejmie informuje, że w I kwartale 1989 r. ukaże się poradnik

ZAGADNIENIA PRAWNE ZAWIERANIA UMÓW W ZAKRESIE INFORMATYKI

Autor: Ryszard PESSEL
Objętość 6 ark. aut., cena ok. 2800,– zł

Praca poświęcona jest problematyce stosunków umownych w dziedzinie informatyki i omawia poszczególne typy umów zawieranych w tym zakresie. Pomyślana jako przewodnik dla informatyków, w wielu zagadnieniach wykracza poza ten schemat. Podejmuje wiele złożonych problemów natury prawnej, występujących w obrocie z zakresu informatyki, m. in. jeden z najważniejszych, dot. ochrony prawnej oprogramowania.
Poradnik przeznaczony jest dla informatyków, inżynierów, ekonomistów i prawników, zajmujących się szeroko rozumianym obrotem w informatyce.

Instytut Ogranizacji Przemysłu Maszynowego
Dział Wydawnictw
ul. Żelazna 87, 00–879 Warszawa
tel. 24-60-61 wew. 258, 297; teleks 813747 IOPM PL

EO/1234/88

Wolfgang D. Ehrenberger

Wolfgang D. Ehrenberger po skończeniu studiów elektronicznych w Technische Hochschule, w Monachium, pracował przez dwa lata w firmie Siemens AG przy projektowaniu komputerowych urządzeń peryferyjnych. Później powrócił do monachijskiej Technische Hochschule, gdzie zajmował się komputerami hybrydowymi, głównie oprogramowaniem. Tam też, w 1971 roku, uzyskał stopień doktora.

Następnie, jako kierownik grupy badaczy pracował nad zastosowaniem systemów komputerowych w elektrowniach jądrowych i systemach sygnalizacji kolejowej. Po przyłączeniu instytutu do nowo powstałej firmy zajmującej się bezpieczeństwem reaktorów (GRS, *Gesellschaft für Reaktorsicherheit),* zajmował się głównie zagadnieniami niezawodności, z uwzględnieniem zagadnień normalizacji. Od 1984 roku, jako profesor inżynierii oprogramowania w Fachhochschule w Fuldzie, prowadzi kilka rodzajów wykładów i jednocześnie ściśle współpracuje z GRS.

Wolfgang Ehrenberger jest członkiem kilku krajowych i międzynarodowych stowarzyszeń naukowych i zawodowych. Był jednym z założycieli European Workshop on Industrial Computer Systems (EWICS) i pierwszym przewodniczącym jego komitetu technicznego nr 7 ds. niezawodności i bezpieczeństwa. Jest członkiem rady naukowej Gütegemeinschaft Software i przewodniczącym grupy roboczej ds. niezawodności oprogramowania przy VDI (Stowarzyszenie Inżynierów Niemieckich).

W IFIP pełni funkcję przewodniczącego grupy roboczej 5.4 ds. komputerowego sterowania procesami. Był przewodniczącym komitetu programowego międzynarodowej konferencji IFIP/IFAC nt. Hardware and Software for Real Time Process Control, która odbyła się w bieżącym roku w Warszawie. **(MK)**

Konferencje

Na przełomie maja i czerwca 1989 r. w Mielnie k. Koszalina planowana jest kolejna Szkoła

Polioptymalizacja w projektowaniu

organizowana przez Wyższą Szkołę Inżynierską i Ośrodek Doskonalenia Kadr NOT w Koszalinie.

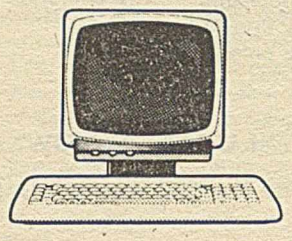
Tematyka Szkoły:

- rola i zadania opt/pol (optymalizacja i polioptymalizacja) w procesie projektowania i konstruowania technicznego,
- podstawowe pojęcia opt/pol na zbiorze nieprzeliczalnym i przeliczalnym (programowanie całkowitoliczbowe),
- metody i algorytmy opt/pol,
- przykłady i demonstracja uniwersalnych systemów komputerowych opt/opl,
- przykłady zadań opt/pol w dynamice i konstrukcji maszyn, w konstrukcjach statycznych i innych zastosowaniach,
- komputerowe wspomaganie projektowania i konstruowania (systemy CAD),
- modele i symulacja komputerowa dla celów projektowania,
- systemy ekspertowe w zadaniach technicznych: komputerowe systemy doradcze w projektowaniu i eksploatacji układów technicznych.

Zgłoszenia przyjmuje:
ZODOK SIMP
ul. Jana z Kolna 38
75-204 Koszalin

Informacji udziela:
Wyższa Szkoła Inżynierska
ul. Racławicka 15
75-560 Koszalin
tel. 278-81 w. 239

Dla użytkowników systemów ODRA i ICL



ZEKOM
ZAKŁAD ELEKTRONIKI KOMPUTEROWEJ

ZEKOM proponuje:

* TERMINAL EKRANOWY MV 2581
przeznaczony do pracy w systemach ODRA 1300 wyposażonych w grupową jednostkę sterującą JSG-7802 jako odpowiednik monitora MERA-7911N
* TERMINAL EKRANOWY MV 2582E
przeznaczony do pracy w systemach ODRA 1300 ICL 1900, ICL 2900, ICL System 4 jako odpowiednik monitora 7181/2 firmy ICL
* MULTIPLEKSER TX 82
przeznaczony do współpracy z terminalami MV 2582E jako 8-kanałowy odpowiednik adaptera QLSA firmy ICL
* ADAPTER LINI TA 42
przeznaczony do współpracy z terminalami MV 2582E jako 4-kanałowy odpowiednik adaptera LSA firmy ICL

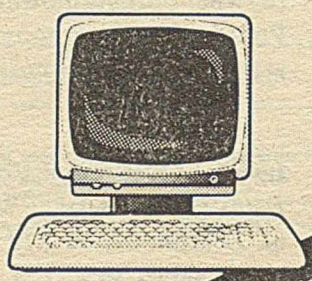
Skr. pocztowa 35, 90-955 Łódź 8, tel. 57-25-83

EO/1088/88

ZAKŁAD ELEKTRONIKI KOMPUTEROWEJ
Skr. pocztowa 35, 90-955 Łódź 8, tel. 57-25-83
ZEKOM

Oferuje użytkownikom systemów MERA-9150 i REDIFON

TERMINAL EKRANOWY
MR 1241



będący funkcjonalnym odpowiednikiem monitora MERA-7951.
Terminal MR 1241 przeznaczony jest do wprowadzania danych do systemu minikomputerowego MERA-9150.

JEŚLI INTERESUJE PAŃSTWA:

* *profesjonalny sprzęt o wysokiej jakości, niezawodny w eksploatacji*
* *sprawny serwis*
* *krótkie terminy dostaw lub dostawy natychmiastowe*

TO WYROBY ZEKOMU SĄ DO PAŃSTWA DYSPOZYCJI

dla użytkowników systemów MERA-9150 i REDIFON kompatybilny z MERA-7951

Data & Knowledge Engineering

Kwartalnik o takim tytule (inżynieria danych i wiedzy), wydawany nakładem wydawnictwa North-Holland, ukazuje się od 1986 roku. Chociaż systemy baz danych i systemy z bazą wiedzy różnią się między sobą, mają także wiele cech wspólnych. Dlatego potrzebne jest wspólne forum wymiany informacji.

Data & Knowledge Engineering służy projektantom, menadżerom i użytkownikom systemów baz danych, systemów ekspertowych oraz systemów z bazami wiedzy. Zakres tematyczny czasopisma obejmuje identyfikowanie, badanie i analizowanie odpowiednich zasad projektowania i efektywnego użytkowania tych systemów. Czasopismo jest poświęcone przeszczepianiu idei i stymulowaniu współpracy między zainteresowanymi instytucjami. Aby osiągnąć ten cel, czasopismo publikuje i rozpowszechnia oryginalne wyniki badań naukowych, osiągnięcia techniczne inżynierii danych, inżynierii wiedzy oraz z pogranicza tych dwóch dziedzin.

Data & Knowledge Engineering kładzie nacisk na następujące zagadnienia:

- **Reprezentacja i manipulowanie danymi i wiedzą**

Konceptualne modele danych, techniki reprezentowania wiedzy, języki i techniki manipulowania danymi i wiedzą.

- **Architektura systemów baz danych, ekspertowych i z bazą wiedzy**

Nowe architektury dla systemów baz danych (z bazą wiedzy lub ekspertowych), techniki projektowania i implementacji, języki i sprzężenia użytkowe, architektury rozproszone.

- **Konstruowanie baz danych i baz wiedzy**

Metodologie i narzędzia do projektowania baz danych i baz wiedzy, metody gromadzenia danych i wiedzy, zagadnienia integralności, poufności i pielęgnowania.

- **Zastosowania, opracowania monograficzne i problemy zarządzania**

Problemy administrowania danymi, praktyka inżynierii wiedzy, zastosowania biurowe i inżynierskie.

Poza artykułami technicznymi w czasopiśmie są także prezentowane raporty konferencyjne, kalendarz wydarzeń, recenzje książek itp.

Redaktorem naczelnym Data & Knowledge Engineering jest prof. Peter P. Chen (Department of Computer Science, Louisiana State University, Baton Rouge, USA), a funkcję regionalnego redaktora dla Europy pełni prof. Reind van de Riet (Department of Mathematics and Computer Science, Free University, Amsterdam, Holandia).

Wybrane artykuły z numerów 3 i 4, 1987

Chao-Chih Yang: On Interpreting and Inferring Propositional Formulas of Data Dependencies in Relational Database.

Artykuł ten dotyczy problemu spełnialności i wnioskowania, obejmującego zależności funkcjonalne i wielowartościowe w relacyjnej bazie danych. W szczególności, zaproponowano dwa niezależne środki do rozwiązania problemu wnioskowania, dotyczące logicznych odpowiedników zależności funkcjonalnych, jak również wielowartościowych. Jeden z tych środków ma postać pary komplementarnych twierdzeń o nierównoważności i równoważności między formułą zdaniową odpowiadającą różnicy, **U–X**, w teorii zbiorów a formułą zdaniową **not(X)**, gdzie **U** jest schematem relacji, a **X** jest podzbiorem **U**. Zastosowanie tych twierdzeń pozwala zapewnić poprawność rozwiązania problemu wnioskowania. Drugim środkiem dla rozwiązania problemu wnioskowania jest wykorzystanie diagramu **Venna**, w celu uproszczenia formuły zadaniowej obejmującej koniunkcje, różnice itp. W artykule podano także i przedyskutowano wskazówki dla konstruowania diagramów **Venna**.

P. Chisholm i in.: Coping with Indefinite and Negative Data in Deductive Databases – A Survey

Jednym z najważniejszych problemów w dzisiejszych systemach baz danych jest ograniczenie nakładane na typy danych, które w tych systemach można reprezentować i którymi można manipulować. Podejmując próbę złagodzenia tego ograniczenia i, w konsekwencji, dopuszczenia szerszej klasy danych (tj. reguł i faktów) oraz mocniejszego zestawu manipulacji, wprowadzono pojęcie dedukcyjnej bazy danych. Jednakże ze względów efektywnościowych postacie typów reguł dozwolonych w dedukcyjnej bazie danych są ograniczone. Artykuł zawiera przegląd wielu prób przejścia do reguł w bazach danych o mniej restryktywnych formach, które pozwalają operować danymi nieokreślonymi (ang. *indefinite*) i zaprzeczeniowymi (ang. *negative*).

Perry Jones, Michael Shave: A Language for Simple Interactive Retrieval from a Database System

Interakcja z systemem bazy danych, w ogólnym wypadku, wymaga pewnego zrozumienia zawartości i struktury bazy danych oraz znajomości odpowiedniego języka zapytań, niezbędnej do zakodowania zapotrzebowania na dane. Te okoliczności stwarzają bariery uniemożliwiające dostęp do bazy danych użytkownikom przypadkowym lub potrzebującym dostępu sporadycznie. Dążąc do przezwyciężenia tego rodzaju ograniczeń, autorzy zbadali możliwość użycia pseudointeligentnego systemu wyszukiwawczego, ukierunkowanego na użytkownika finalnego. System ten został tak zaprojektowany, aby nie był zależny od żadnego konkretnego systemu zarządzania, chociaż uznano, że najbardziej właściwa dla takiego systemu byłaby relacyjna struktura danych. Prototypowa wersja systemu wyszukiwawczego została przygotowana do działania jako nadbudowa systemu zarządzania relacyjną bazą danych **RAPORT** oraz systemu relacyjnej bazy danych **SQL** firmy IBM. Jako rezultat doświadczeń z tym systemem powstał adaptowalny język ułatwiający inteligentną współpracę między użytkownikiem finalnym a systemem zarządzania bazą danych.

W. RZECZKOWSKI

Nowe książki

dokończenie ze s. 6

Dwa ostatnie rozdziały, szósty i siódmy, zawierają omówienie metod zarządzania siecią – od ułożenia kabli aż do diagnostyki – i doświadczeń eksploatacyjnych, m. in. porównanie z siecią PC Network, porównanie pracy stacji opartych na komputerach PC/XT i PC/AT oraz wskazówki praktyczne. Książkę kończy dodatek stanowiący wykaz ostatnio wprowadzonych na rynek wyrobów firmy IBM, służących do konfigurowania tej sieci.

G. Held, Komunikacja z komputerami serii IBM PC. Wiley and Sons, Chichester (UK), 1988, ISBN 0471--916676

W książce omówiono, w sposób jasny i przystępny, możliwości komunikacyjne komputerów IBM PC i kompatybilnych, poczynając od pojęć podstawowych, przez zagadnienia sprzętu i oprogramowania, aż do struktur sieciowych i kryteriów wyboru usług. Poszczególne rozdziały zestawiono metodą modularną, aby można było je czytać w możliwie dowolnej kolejności. Rozdział dotyczący sprzętu zawiera m.in. omówienie płyt adapterów komunikacyjnych dla PC, a rozdział dotyczący oprogramowania – opis właściwości programowych modułów komunikacyjnych dla tych komputerów. W dwóch następnych rozdziałach przedstawiono technologię i realizację sieci opartej na komutacji pakietów, omawiając zapewniane przez nią usługi, oraz – rzeczywistą sieć opartą na komputerach Systemu/3X i stacjach 3270. W końcowych rozdziałach zawarto wiele szczegółowych uwag o projektowaniu sieci lokalnych dla komputerów osobistych i metodach ich integracji z sieciami łączności istniejącymi już w przedsiębiorstwach. Podane przykłady praktyczne ilustrują użycie sprzętu i oprogramowania w sposób dostateczny na tyle, aby ułatwić czytelnikom samodzielny wybór odpowiednich rozwiązań.

(J. ZAL.)

Tucholski A.: NETBIOS – zasada działania i sposób użytkowania (1)
INFORMATYKA 1988, nr 11–12, s. 1
Pierwsza część charakterystyki standardowego zbioru procedur NETBIOS zapewniającego sprzętowo niezależne sprzężenie między adapterem sieci komputerowej a systemem operacyjnym DOS.

Tucholski A.: NETBIOS – operation principle and application method (1)
INFORMATYKA 1988, No. 11–12, p. 1
First part of standard procedures set characteristics, which, assures hardware independent interface between computer network adapter and the DOS operating system.

Tucholski A.: NETBIOS – Operationsgrundsatz und Anwendungsweise (1)
INFORMATYKA 1988, Nr. 11–12, S. 1
Erster Teil einer Charakteristik von NETBIOS, einer standarden Prozedurmenge, die eine hardwareunabhängige Schnittstelle zwischen Rechneradapter und dem DOS-Betriebssystem sichert.

Lamport L.: Uproszczone specyfikowanie systemów współbieżnych (2)
INFORMATYKA 1988, nr 11–12, s. 7
Druga część prezentacji nowej metody ułatwiającej specyfikowanie systemów współbieżnych.

Lamport L.: Simplified concurrent system specyfying (2)
INFORMATYKA 1988, No. 11–12, p. 7
Second part of presentation of a new method, which makes concurrent systems specifying easy.

Lamport L.: Vereinfachte Spezifizierung der parallen Systeme (2)
INFORMATYKA 1988, Nr. 11–12, S. 7
Zweiter Teil einer Präsentation von neuer Methode, die Spezifizierung der parallelen Systeme erleichtert.

Mykowiecka A., Ostaszewska A.: Rozmowa z komputerem
INFORMATYKA 1988, nr 11–12, s. 11
Charakterystyka problemu oraz obecnych rozwiązań w dziedzinie komunikacji z komputerem w języku naturalnym.

Mykowiecka A., Ostaszewska A.: Conversation with computer
INFORMATYKA 1988, No. 11–12, p. 11
Characteristics of the problem and actual solutions in communication with computer in natural language.

Mykowiecka A., Ostaszewska A.: Gespräch mit Computer
INFORMATYKA 1988, NR. 11–12, S. 11
Eine Charakteristik des Problems und der heutigen Lösungen im Bereich der Kommunikation mit Computer in natürlicher Sprache.

Gizdoń H., Pawlak A., Wrona W.: Język opisu sprzętu VHDL – podstawowe mechanizmy (1)
INFORMATYKA 1988, nr 11–12, s. 15
Pierwsza część szczegółowej charakterystyki języka opisu sprzętu VHDL, zawierająca rozwiązania wersji 7.2 tego języka.

Gizdoń H., Pawlak A., Wrona W.: VHDL hardware description language – basic mechanisms (1)
INFORMATYKA 1988, No. 11–12, p. 15
First part of detailed characteristics of the VHDL hardware description language, which includes 7.2 version solutions of this language.

Gizdoń H., Pawlak A., Wrona W.: VHDL-Hardwarebeschreibungssprache – die Grundmechanismen (1)
INFORMATYKA 1988, Nr. 11–12, S. 15
Erster Teil einer detailierten Charakteristik von VHDL-Hardwarebeschreibungssprache, der die Lösungen von Version 7.2 dieser Sprache enthält.

Czachórski T., Kowalówka M., Szczerbiński Z., Wilk A.: AMOK – mikrokomputerowy system doradczy projektanta i analityka systemów komputerowych
INFORMATYKA 1988, nr 11–12, s. 19
Charakterystyka pakietu programowego AMOK do modelowania i oceny działania systemów komputerowych. Pakiet ten jest przeznaczony do stosowania na mikrokomputerach klasy IBM PC.

Czachórski T., Kowalówka M., Szczerbiński Z., Wilk A.: AMOK – a microcomputer expert system for designer and analyst of computer systems
INFORMATYKA 1988, No. 11–12, p. 19
Characteristics of the AMOK program package for modelling and evaluation of computer systems operation. The package is appropriated for application on microcomputers of the IBM PC class.

Czachórski T., Kowalówka M., Szczerbiński Z., Wilk A.: AMOK – eine Mikrorechnerexpertensystem für Projektanten und Analytiker von Rechnersystemen
INFORMATYKA 1988, Nr. 11–12, S. 19
Eine Charakteristik des Programmpakets AMOK für Modellierung und Bewertung der Rechnersystemwirkung. Dieser Paket ist für Anwendung auf Mikrorechnern der IBM PC-Klasse bestimmt.

Fuglewicz P., Topczyński R.: DAGGER – program utrzymania plików
INFORMATYKA 1988, nr 11–12, s. 23
Charakterystyka nowego programu narzędziowego dla komputerów rodziny Mera 60/660, eliminującego konieczność korzystania w tradycyjnej formie z większości poleceń systemowych.

Fuglewicz P., Topczyński R.: DAGGER – a program for file maintenance
INFORMATYKA 1988, No. 11–12, p. 23
Characteristics of a new utility program for Mera 60/660 computer family, which eliminates usage of majority of system commands applied in traditional form.

Fuglewicz P., Topczyński R.: DAGGER – ein Programm für Dateipflege
INFORMATYKA 1988, Nr. 11–12, S. 23
Eine Charakteristik von neuen Utilityprogramm für Mera 60/660-Rechnerfamilie, der die Mehrheit traditionellen Form angewendeten Systembefehle eliminiert.

Cellary W., Kręglewski J., Maćkowiak R.: System komputerowy Elwro 800 (2)
INFORMATYKA 1988, nr 11–12, s. 26
Druga część charakterystyki systemu mikrokomputerowego Elwro 800, zawierająca omówienie jego modułów funkcjonalnych.

Cellary W., Kręglewski J., Maćkowiak R.: Elwro 800 microcomputer system (2)
INFORMATYKA 1988, No. 11–12, p. 26
Second part of characteristics of the Elwro 800 microcomputer system, which includes discussion of its functional modules.

Cellary W., Kręglewski J., Maćkowiak R.: Elwro 800-Mikrorechnersystem (2)
INFORMATYKA 1988, Nr. 11–12, S. 26
Zweiter Teil einer Charakteristik von dem Elwro 800-Mikrorechnersystem, der eine Besprechung seiner funktionellen Module umfasst.

Sysło M.M.: Maszyny i algorytmy równoległe (2)
INFORMATYKA 1988, nr. 11–12, s. 30
Druga część przeglądu zrealizowanych modeli równoległych systemów komputerowych oraz zaprojektowanych dla nich algorytmów, zawierająca charakterystykę maszyn typu MIND.

Sysło M. M.: Parallel machines and algorithms (2)
INFORMATYKA 1988, No. 11–12, p. 30
Second part of the survey presenting realized models of parallel computer systems and for them designed algorithms, which includes characteristics of the MIND-type machines.

Sysło M. M.: Parallele Maschinen und Algorithmen (2)
INFORMATYKA 1988, Nr. 11–12, S. 30
Zweiter Teil einer Übersicht von realisierten Modellen der parallelen Rechnersysteme, der eine Charakteristik von Maschinen des MIND-Types enthält.

Bielecki J.: Turbo C. Przerwania, sygnały, powroty
INFORMATYKA 1988, nr 11–12, s. 35
Praktyczne wskazówki i przykłady stosowania niektórych rozwiązań języka programowania Turbo C na mikrokomputerach klasy IBM PC.

Bielecki J.: Turbo C. Interrupts, Signals, Returns
INFORMATYKA 1988, No. 11–12, p. 35
Practical advices and examples for application of some solutions of the Turbo C language on microcomputers of IBM PC class.

Bielecki J.: Turbo C. Unterbrechungen, Signale, Rükkehre
INFORMATYKA 1988, Nr. 11–12, S. 35
Praktische Hinweise und Beispiele der Anwendung von manchen Lösungen der Turbo C-Programmiersprache.

Terminologia systemów operacyjnych (3)

Rozważania prowadzone w dwóch poprzednich odcinkach dotyczyły poleceń wydawanych do systemu operacyjnego – i terminów pokrewnych. Jak wiadomo, polecenia wydawane przez użytkownika są przyjmowane przez specjalny program obsługujący komunikację z użytkownikiem, zwany **interpreterem poleceń** (ang. *command interpreter*). Choć twierdzi się, że nazwa interpreter jest niepoprawnym uproszczeniem i należałoby mówić interpretator (tj. program, który interpretuje) nie sądzę, aby przy znacznym stopniu zadomowienia tego uproszczonego terminu udał się odwrót od jego używania. Jest tak tym bardziej, że nazwa interpretator uwzględnia w większym stopniu element ludzki odpowiedniej czynności a interpreter – element maszynowy, a takie rozróżnienie jest nawet pożądane.

Pewien szczególny rodzaj interpretera poleceń, używany w systemie Unix, nazywa się **shell,** co najdokładniej tłumaczy się na język polski jako **powłoka.** Właśnie termin **powłoka** najwymowniej ilustruje fakt, że system operacyjny to nie tylko widoczna na zewnątrz część przyjmująca polecenia. Oczywiście, powłoka nie wykonuje poleceń – zaledwie je rozpoznaje, kierując odpowiednie wywołania do wewnętrznej części systemu operacyjnego, tzw. **jądra** (ang. *kernel, nucleus*). Powłoka otacza więc jądro. Jądro bywa nazywane najróżniejszymi terminami angielskimi, np. w systemie RSX-11 nosi nazwę *executive,* co najbardziej naturalnie tłumaczy się jako **egzekutor.**

Komunikacja między powłoką a jądrem odbywa się za pomocą wywołań systemowych, czyli tzw. dyrektyw. Oczywiście, są też możliwe bezpośrednie wywołania jądra z programów użytkowych. Istotne jest jednak to, że jądro rzadko jest pojedynczym programem, a nawet jeśli takim jest, to składa się z wielu współbieżnie działających elementów zwanych **procesami** (ang. *process*) lub **zadaniami** (ang. *task*). Ponieważ procesy (zadania) systemu operacyjnego współdzielą procesor wraz z procesami użytkowymi, musi istnieć pewna hierarchia zapewniająca wszystkim procesom właściwy dostęp do procesora. Odbywa się to przez przypisanie procesom ściśle określonych stanów świadczących o aktualnych warunkach, w których się znajdują. Ponieważ każdy system operacyjny jest zrealizowany inaczej, nie ma reguły na to, w ilu i w jakich stanach mogą znajdować się jego procesy (zadania). Mogą to być, na przykład, stany uśpienia, gotowości, aktywności i wiele innych.

Zanim przejdę do nazw samych stanów, chciałbym zwrócić uwagę na pewną zasadę poprawnego nazywania stanów, która rzadko jest przestrzegana. Otóż, na pytanie „W jakim stanie jest obiekt?" odpowiada się na ogół za pomocą przymiotnika, np. jest w złym stanie, w dobrym stanie, w opłakanym stanie itp., ogólnie „Jest w stanie ⟨takim a takim⟩". Kształt odpowiedzi wynika z formy pytania, ponieważ zaimek „jaki" wskazuje na jakość przedmiotu. Jednakże w wypadku pytania o aktualny stan procesu chodzi nam nie o ogólną jakość stanu, ale o sam stan wewnętrzny. Dlatego powinniśmy zapytać, w którym z dopuszczalnych stanów jest nasz proces? Wtedy nie można odpowiedzieć, że proces jest w stanie uśpionym, gotowym, aktywnym itd., lecz należy użyć przydawki dopełniaczowej – proces jest w stanie uśpienia, gotowości, aktywności itp. Pytanie „W jakim stanie jest proces?" należy więc traktować jako uproszczoną formę pytania „W którym z dopuszczalnych stanów jest proces?"

Wyróżnionymi stanami procesu mogą być, na przykład, następujące:
- nieobecność (ang. *non-existence*),
- uśpienie (ang. *hibernation, dormancy*),
- gotowość (ang. *readiness*),
- aktywność (ang. *activation*),
- bieg (ang. *running*),
- zablokowanie (ang. *blocking*),
- oczekiwanie (ang. *waiting*),
- zawieszenie (ang. *suspension, hanging-up*),
- załatwianie (ang. *pending*),
- spełnienie (ang. *completion*),
- zakończenie (ang. *termination*).

Przykładowo wymienionych kilkanaście wariantów oczywiście nie wyczerpuje wszystkich stanów, w których może znaleźć się proces, choć w większości systemów operacyjnych ich liczba jest ograniczona do kilku.

Zwróćmy uwagę, że podana zasada nazewnictwa stanów nie zabrania używania przymiotników, ale w innej konstrukcji językowej. Na pytanie, w jakim stanie jest proces, można odpowiedzieć: proces jest nieobecny (ang. *non-existent*), uśpiony (ang. *hibernated, dormant*), gotowy (ang. *ready*), aktywny (ang. *active*), biegnący (ang. *running*), zablokowany (ang. *blocked*), oczekujący (ang. *waiting*), zawieszony (ang. *suspended, hung-up*), niezałatwiony (ang. *pending*), spełniony (ang. *completed*), zakończony (ang. *terminated*) itp.

W tym miejscu trzeba też zasygnalizować, że termin *run* rozumiany jako polecenie do systemu operacyjnego (a więc widoczne z zewnątrz) oznacza – **wykonaj**, ale w odniesieniu do stanu wnętrza systemu jest rozumiany inaczej (wyraz ma więc inne znaczenie) i po polsku może być przetłumaczony odmiennie. Nie można też polecenia *run* nazywać uruchomieniem (choć w znaczeniu potocznym jest to poprawne i zrozumiałe), ponieważ w informatyce **uruchamianie** jest czymś zupełnie innym i jak wiadomo, odpowiada angielskiemu terminowi *debugging.*

W doborze pozostałych odpowiedników należy uwzględnić też inne konflikty, np. odróżnić w języku polskim stany *completion* i *termination,* czy *waiting* i *pending* itp. Dodatkowy kłopot w poprawnym nazywaniu stanów po polsku może polegać na tym, że w języku angielskim zarówno imiesłów czynny jak i rzeczownik odsłowny jest tym samym wyrazem zakończonym na -ing, np. *waiting* (oczekujący, oczekiwanie).

Niewielka grupa terminów dotyczy dynamicznego powoływania procesów przez inne procesy. Proces powołujący nazywa się po angielsku *parent process* lub *father process,* a proces powoływany – *child process* lub *offspring process.* Polskie terminy, cechujące się najmniejszym zabarwieniem emocjonalnym, związanym z powielaniem procesów, to odpowiednio – **proces macierzysty** (lub przodek) i **proces potomny** (lub potomek). Procesy mające ten sam proces macierzysty nazywa się najczęściej *sibling processes,* co można przetłumaczyć jako **procesy siostrzane.** Samo tworzenie procesów jest bardzo różnie nazywane po angielsku i zależy od abstrakcyjnego modelu opisu procesów. Częściej spotykanymi terminami angielskimi są: *fork* i *spawn,* co można przetłumaczyć jako **rozwidlanie** i **rozmnażanie.**

Oczywiście, najbardziej fascynujące, choć już nie dla terminologów, a dla programistów, jest takie zaprojektowanie systemu operacyjnego, aby w sposób płynny i bezkonfliktowy wywoływać i uwzględniać zmiany stanów procesów. Główny program służący do tego celu jest określany po angielsku mianem *scheduler* lub *dispatcher* (często są to dwa różne programy). Rola programu zwanego *scheduler* polega na regulowaniu dostępu procesów do procesora, a więc – szeregowaniu procesów, dlatego po polsku nazywa się go najczęściej **programem szeregującym.** Rola programu zwanego *dispatcher* jest analogiczna, choć widziana od strony procesora polega na dokonywaniu przydziału procesora tym zadaniom, które są w danej chwili uprawnione do posiadania procesora. Taki program dysponuje więc procesorem i dlatego nazywa się po polsku **dyspozytorem.**

Gospodarowanie procesorem jest ściśle powiązane z gospodarowaniem pamięcią. Również w tej dziedzinie nie wszystkie terminy są utrwalone, a nawet ustalone. Przykładowo, pewne obszary pamięci (operacyjnej lub dyskowej) o ściśle zdefiniowanym przeznaczeniu

dokończenie na s. 49

Ceny ogłoszeń

Od 1 lipca 1987 r. obowiązują następujące ceny materiałów reklamowych publikowanych na łamach INFORMATYKI:

Ogłoszenia

- ogłoszenia czarno-białe, artykuły reklamowe i informacje naukowo-techniczne (biuletyny) zależnie od objętości: cała strona – 50 tys., 3/4 str. – 45 tys., 2/3 str. – 40 tys., 1/2 str. – 35 tys., 1/3 str. – 30 tys., 1/4 str. – 25 tys., 1/8 str. – 20 tys., poniżej 1/8 str. – 200 zł za 1 cm^2,
- ogłoszenia drobne (zależnie od liczby słów) jedno słowo – 50 zł

Dodatki do ceny podstawowej:

- za każdy dodatkowy kolor + 30%,
- za każdy specjalny kolor (nie wynikający z podstawowych kolorów) + 30%,
- za pełny kolor (grafika wielobarwna, zdjęcia kolorowe) + 120%,
- za zamieszczenie ogłoszenia na I lub IV stronie okładki + 100%,
- za zamieszczenie ogłoszenia na II i III stronie okładki + 50%

Zniżki

dotyczą ogłoszeń – całkowitych powtórzeń

- za ogłoszenia 3-5-krotne – 10%,
- za ogłoszenia 6-10-krotne – 20%,
- za ogłoszenia 11-krotne i powyżej – 30%

- za artykuły i wkładki reklamowe wykonane przez zleceniodawcę – 40%,
- za biuletyny i bloki reklamowe – 60%

W uzasadnionych wypadkach stosuje się zniżki specjalne dla ogłoszeń nie będących powtórzeniami – za zgodą Dyrektora – Naczelnego Redaktora Wydawnictwa NOT SIGMA. Ponadto Biuro Ogłoszeń świadczy usługi w zakresie wykonywania zdjęć czarno-białych i barwnych oraz nadbitek wkładek reklamowych.

Ceny wkładek

- wkładka 2 str. o formacie A4
  nakład do 500 egz. 20 tys. zł
  nakład 500-1000 egz. 35 tys. zł
- wkładka 4 str. o formacie A4
  nakład 500 egz. 40 tys. zł
  nakład 500-1000 egz. 70 tys. zł

Przy wkładkach o nakładzie powyżej 1000 egz. stosuje się wielokrotność powyższych cen a dodatki za kolory oblicza się jak dla ogłoszeń.

Ogłoszenia przyjmowane są przez:
Dział Ogłoszeń i Reklamy WCiKT NOT SIGMA
ul. Świętojerska 5/7, 00-236 Warszawa
adres do korespondencji: skrytka pocztowa 1004, 00-950 Warszawa
telefony: 31-93-65 lub 31-22-21 w. 196 i 291

Warunki prenumeraty na lata 1988–1989

Prenumeratorzy zbiorowi – jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat wyłącznie na blankiecie „wpłata-zamówienie" (jest to „polecenie przelewu" rozszerzone dla potrzeb Wydawnictwa o część dotyczącą zamówienia).

Blankiety te będą dostarczane dotychczasowym prenumeratom przez Zakład Kolportażu. Nowi prenumeratorzy otrzymają je po zgłoszeniu zapotrzebowania (pisemne lub telefoniczne) w Zakładzie Kolportażu.

Prenumeratorzy indywidualni – osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty. Wpłacać należy na konto: NBP III Oddział Warszawa 1036-7490-139-11.

Prenumerata ulgowa – przysługuje wyłącznie osobom fizycznym – członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły. Sposób zamawiania prenumeraty ulgowej jest taki sam jak prenumeraty indywidualnej. W prenumeracie ulgowej można zamówić tylko po jednym egzemplarzu każdego czasopisma.

Uwaga! Miesięcznik „Aura" może być zamawiany w prenumeracie ulgowej również przez uczniów szkół ogólnokształcących.

Prenumerata ze zleceniem wysyłki za granicę – zamawia się tak jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy.

Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Wpłaty na prenumeratę przyjmowane są w terminach:

- do 10 listopada na każdy kwartał, I i II półrocze oraz cały rok następny,
- do 28 lutego na II, III i IV kwartał oraz II półrocze,
- do 31 maja na III i IV kwartał oraz II półrocze,
- do 31 sierpnia na IV kwartał.

Zmiany w prenumeracie można zgłaszać pisemnie tylko w wyżej wymienionych terminach.

Informacji o prenumeracie udziela – Zakład Kolportażu Wydawnictwa NOT SIGMA (ul. Bartycka 20, 00-716 Warszawa), skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 wew. 248, 249, 293, 297, 299 lub 40-30-86 i 40-35-89.

Egzemplarze archiwalne czasopism – można nabyć za gotówkę w Klubie Prasy Technicznej, Warszawa ul. Mazowiecka 12 (tel. 27-43-65), lub zamówić pisemnie. Zamówienia na egzemplarze archiwalne czasopism przyjmuje: Zakład Kolportażu, Dział Handlowy, 00-950 Warszawa, skr. poczt. 1004 (tel. 40-37-31), na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

Cena w 1988 roku

(dotyczy numerów 7-12)

CENA		Prenumerata normalna		Prenumerata ulgowa (bez zmiany)	
normalna	ulgowa (bez zmiany)	kwartalna	półroczna	kwartalna	półroczna
250 zł	50 zł	750 zł	1500 zł	150 zł	300 zł

Cena w 1989 roku

miesięczna		kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
300 zł	60 zł	900 zł	180 zł	1800 zł	360 zł	3600 zł	720 zł